An Integration Framework for Managing Rich Organisational Process Knowledge

Stephen T. Polyak

Ph.D.
University of Edinburgh
1999

Abstract

The problem we have addressed in this dissertation is that of designing a pragmatic framework for integrating the synthesis and management of organisational process knowledge which is based on domain-independent AI planning and plan representations. Our solution has focused on a set of framework components which provide methods, tools and representations to accomplish this task.

In the framework we address a lifecycle of this knowledge which begins with a methodological approach to acquiring information about the process domain. We show that this initial domain specification can be translated into a common constraint-based model of activity (based on the work of Tate, 1996c and 1996d) which can then be operationalised for use in an AI planner. This model of activity is ontologically underpinned and may be expressed with a flexible and extensible language based on a sorted first-order logic. The model combines perspectives covering both the space of behaviour as well as the space of decisions. Synthesised or modified processes/plans can be translated to and from the common representation in order to support knowledge sharing, visualisation and mixed-initiative interaction.

This work united past and present Edinburgh research on planning and infused it with perspectives from design rationale, requirements engineering, and process knowledge sharing. The implementation has been applied to a portfolio of scenarios which include process examples from business, manufacturing, construction and military operations.

Acknowledgements

The author is sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-96-1-0348 – an AASERT award monitored by Dr. Abe Waksman and associated with the O-Plan project F30602-97-1-0022. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of DARPA,AFOSR or the U.S. Government.

My biggest thanks goes to my primary supervisor, Prof. Austin Tate, who has not only enabled this work through the AASERT award but has mentored and encouraged me over the last three years and made this a very positive and enjoyable experience. I am very grateful for his time and attention and his continued, enthusiastic involvement throughout this thesis work. I am also thankful for the many project/professional contacts that Austin made available to me which led to outstanding collaboration opportunities.

I'd also like to thank my secondary supervisor, Dave Roberston, not only for his valuable knowledge, perspectives and insights, but also for bringing me into his research group, Software Systems and Processes (SSP). My experience with the SSP group exposed me to many research interests, provided me with a forum for discussing research ideas, and proved to be lots of fun too.

Thanks to the members of the O-Plan team for all of the support and for insights gained from involvement at the O-Plan meetings. Special thanks to Peter Jarvis for discussions about AI planning, for the many comments on drafts of both my papers and my thesis and for acting as an additional supervisor during this work. Thanks also to Gerhard Wickler for the friendship as well as the discussions about various research issues. Thanks also to Brian Drabble who unfortunately had to accept a great opportunity and head off to Oregon, but who also provided me with some good ideas and assistance. Thanks to Stuart Aitken for the assistance on the NIST PSL work.

Thanks to the PIF members for the weekly discussions and for the opportunity to be involved in the project. A big thanks goes to Jintae Lee, Mike Uschold, Michael Gruninger, Chris Menzel and Hee-Woong Kim for many interesting conceptual arguments and knowledge sharing discussions and also to Pat Hayes for his feedback and interest on the temporal mapping work. Thanks to the NIST PSL members again for the weekly discussions and insights into manufacturing knowledge. This included some of the PIF members as well but also a big thanks to Craig Schlenoff, Amy Knutilla, and Steve Ray. Thanks to the core SPAR members as well for the many discussions and for opportunities to exchange information on approaches to shared plans. Thanks goes to especially to Adam Pease, Karen Myers, and Bill Swartout.

Thanks to the Friday pub night crew who met during October 1996 to October 1997. I really enjoyed the many pints, bags of crisps, and discussions spent over both of them. Special thanks goes to some of the friends that me and my wife made there: Ceri Stagg, Kyle McCluskey, Stephen Cresswell, and Andrew Tuscon. Thanks to Ceri for a Christmas away from home, to Kyle for his French cooking skills, to Stephen for helping to organise the 1997 Robert Burns Supper, and to Andrew for helping co-organise the 1997 Advanced Topic Lecture Series: Planning and Scheduling.

Thanks also to the friends we made through the University. Thanks to Juergen and Anja for the many trips around Scotland and for the all of the fun times. Thanks also to Faulds Advertising, not only for providing Amy with an interesting and fun job, but also for all of the people we met there as well including our good friends Matthew and Emma.

A more personal thanks to the members of the SSP group who were lots of fun to meet with for the weekly meetings and lunch. Brazilians: Daniela, Virginia, Joao, Alberto, Edjard; Non-braziliains: Yannis, Renaud, Jessica, Jon, Peter, Nam and Stefan. Thanks also to Daniela, Ze, Virginia, Elias, Joao, Jason and Fernada for the friendship and fun times.

I'd also like to thank some of the people who made this opportunity possible before I even arrived at Edinburgh. Thanks to Dr. Steven Lytinen at DePaul University in Chicago for getting me interested in AI during my M.Sc. degree and for writing me a recommendation for this Ph.D. work. Thanks also to Dr. Clark Elliott (DePaul University) for sharing information about his work with me, for encouraging me to pursue the Ph.D. and also for recommending me as well.

I'd like to thank my family as well as our friends from home who have been so supportive while we were away from them. Thanks for coming out to Scotland to visit, for the many packages, cards, notes, and letters. Thanks for encouraging me throughout this time and for helping us to manage all of the various details involved in rooting up ones life and moving from country to country.

Finally, and most importantly of all, I'd like to thank my wife, Amy, who has been the real person who has made this all possible. Amy very selflessly set aside her own career to allow us to take this opportunity. She has managed many of the details and supported me throughout our time here.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Steve Polyak
Edinburgh
June 25, 1999

Acronyms/Glossary

Term	**Meaning**
ACP3	Air Campaign Planning Process Panel
ARPI	DARPA/Air Force Research Laboratory (Rome) Planning Initiative
BPR	Business Process Reenginering
CDE	Common Domain Editor
CPA	Common Process Assistant
CPE	Common Process Editor
CPF	Common Process Framework
CPL	Common Process Language
CPM	Common Process Methodology
CPO	Common Process Ontology
CPT	Common Process Translator
CTD	Combined Thread Diagram
DCD	Data Composition Diagram
HTN	Hierarchical Task Network
IDL	Interface Definition Language
<I-N-OVA>	Issues, Nodes, Orderings/Variables/Auxiliary Constraint Model of Activity
ITD	Isolated Thread Diagram
KADS	Knowledge Analysis and Documentation System
MDMP	Military Decision-Making Process
NIST	National Institute of Standards and Technology
OPO	O-Plan Plan Output Format
PIF	Process Interchange Format
PSL	Process Specification Language
SPAR	Shared Planning and Activity Representation
T/R	Transmit and Relay
TED	Tabular Entry Diagram
TIE	Technology Integration Experiment
TQM	Total Quality Management
UML	Unified Modeling Language
VBD	Viewpoint Bubble Diagram
VSD	Viewpoint Structure Diagram

Contents

Chapter 1

Introduction

The aim of this thesis is to answer the question: ***How can we improve the methodology of synthesising and managing organisational process knowledge?*** *Our solution involves the development of a pragmatic framework based on research into AI planning and plan representations which can be used to integrate rich process-related knowledge. We begin our presentation of this work by introducing some motivations and background information before defining the problem and over-viewing our research methods.*

An organisation is a collection of people who perform some activities with the intent of achieving shared objectives. Organisations include companies, sets of companies, governments, military forces, sports teams, standards bodies, departments, business units, clubs, squads, and so on. As our description suggests, one common concept in all organisations is a group of related activities. The term often used to refer to such a concept is *process*.

The effectiveness, and consequently the value, of an organisation is often linked to the quality of its processes. This insight is reflected by programmes aimed at identifying and improving them, e.g. Total Quality Management (TQM) [Cartin, 1993] and Business Process Reengineering (BPR) [Davenport, 1993, Hammer and Champy, 1994]. Many organisations have switched their focus from narrowly defined individual tasks to holistic processes. In strategies such as Knowledge Management [Davenport and Prusak, 1998], companies are being encouraged to represent and manage knowledge of their processes just as they would be expected to represent knowledge of their customers and their orders.

Process representation can be quite cumbersome and time-consuming depending on the amount of detail required. On the one hand organisations may choose to utilise simple representations which highlight some of the main process aspects (cf. Unified Modelling Language's (UML) activity diagrams [Booch *et al.*, 1998]). The advantage is that less time and effort is required, but the result may severely limit the applicability of this work. On the other hand, organisations may need to specify numerous detailed elements which collectively serve to constrain the possible interpretation and enactment of process steps. Both approaches may be valid, but it depends on the organisational context and the intended use of this knowledge.

Many organisations have looked to technological approaches to aid in the process of creating, critiquing, communicating, and collaborating with process knowledge (cf. [Davidow and Malone, 1992]). One technology field which has had a long history of working with the representation of activities and processes is Artificial Intelligence (AI). Specifically, a sub-area of AI known as AI planning [Allen *et al.*, 1990, Weld, 1994, Kambhampati, 1997, Weld, 1999] encompasses a rich set of representations and techniques for reasoning with plans and processes.

In this thesis, we present original work connecting current perspectives on AI-based models of activities, processes and plans with past research on eliciting plan/process requirements and the provision of "intelligent" planning tools. Our approach also incorporates ideas from requirements and ontological engineering, design rationale and knowledge sharing and unifies this into a coherent research framework aimed at realizing the four areas of process management support outlined in [Tate, 1996c]: knowledge acquisition, user communication, analysis, and system manipulation.

1.1 Background to the Research

> *At this point we have introduced some of the context for this research. We are interested in providing support for managing organisational process knowledge by utilising research from AI planning. Process management implies several tasks. For example, new process configurations may need to be synthesised. This corresponds to the generative AI planning task. In this section we provide some background information on process know-*

ledge, planning and open research issues before we proceed to describing the research problem and discussing our hypotheses.

What is organisational process knowledge, or more to the point, what does it entail? In this thesis we adopt the perspective that organisational processes are designed artifacts (e.g. similar to the project management approach in [Petrie *et al.*, 1999]). We argue that processes are designed in ways which are analogous to the design of products or goods. This gibes with our intended application of AI planning. In the planning literature, Tate presents a conceptualisation of *design* which simply states that it is a "set of *constraints* on the relationships between the entities involved in the artifact" [Tate, 1996d]. This definition is then specialised for a *plan* by naming these entities as agents, their purposes, and their behaviour. Thus, an individual organisational process equates to a designed plan artifact.

In the literature review in Chapter 2, we chart the development of AI planning from its primordial roots to its modern constraint-based approaches. An early generic model of domain-independent AI planning and plan execution, reproduced in Figure 1.1, can be used to understand the relationship between the lifecycle steps of designing organisational processes and performing them.

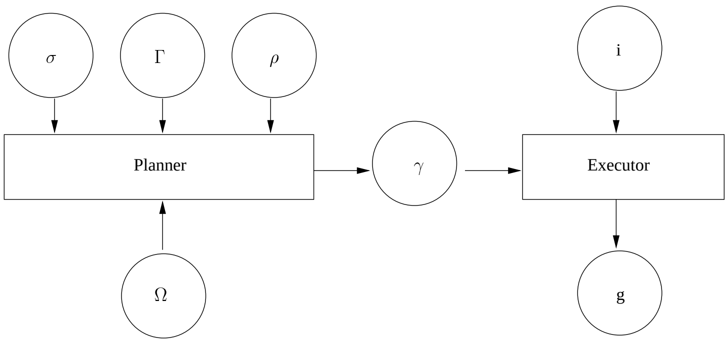

Figure 1.1: A model of the AI planning approach [Genesereth & Nilsson, 1988, pp. 286]

We can see that the design process (denoted by the planner) takes four unique inputs: $\rho, \sigma, \Gamma, \Omega$. Input ρ indicates a description of the goal. This represents our intention of what the process is going to achieve. For example, if the process is intended to validate a customer's credit, we might specify that the goal is to either know whether the credit is sufficient or not. σ is a description of the initial state. For our example, this might assert something about the fact that we would initially need to know the customers id or credit information and the order amount. Γ is a set of “action designators”. The elements in this set denote possible actions in the domain. In the credit checking domain, or more generally, an order processing domain, this might involve actions such as “validate via electronic credit check” or “validate via voice credit check”. We might suspect that the effects of both actions are the same, but the conditions on their performance may be different. This is clarified by the Ω input which is a database of information that describes available actions and state descriptions.

The output of the design process is γ. Given our example, γ is the organisational process for enacting a credit check. When it is time to perform this process γ becomes an input, along with the description of some state, i, which *satisfies* the initial description, σ. Performing this action brings about a state, g, which correspondingly satisfies ρ. If we were using a rigorous logic for representing these elements, we would be able to *prove* from Ω that our process (γ) achieves a valid credit check goal state (ρ) when executed in the initial state (σ), which is expressed as $\Omega \models (\rho(result(\gamma, \sigma)))$ if we were using the situation calculus (see Sections 2.3.1.1 and 2.4).

1.1.1 Plans and Plan Domains

One of the important distinctions outlined in Figure 1.1 is between a plan, or organisational process, and the domain for which this process was designed. In a knowledge-level review of AI planning [Valente, 1995], Valente labelled these the plan description (γ) and the world description (Γ, Ω). The bulk of AI planning research has focused on methods of developing or generating plan descriptions. Only recently though have researchers begun to address principled methods for developing world descriptions (see Section 2.3.7). World descriptions add value to plans in many ways. This value goes beyond its role in generating plans and extends to aspects of plan management. Simple approaches to modelling organisational processes typically omit explicit world descriptions and go straight to specifying process steps.

1.1.2 Common Process Approaches

We believe that the application of both the domain-independent model of AI planning and the representations of rich world and plan descriptions to the management of organisational processes cuts across industries. The elements of this approach could be made *common* to areas as diverse as manufacturing, business, or military processes. In fact, during the period of research for this dissertation the author participated in efforts designed to standardise representations of processes in order to support the sharing of plan/process knowledge. The research from AI planning was one of the major sources of input to these efforts.

We conducted significant work on the Process Interchange Format (PIF) project [Lee *et al.*, 1998a], in participation with the other PIF working group members, to help define and understand issues related to managing business-related process knowledge. For example, we developed and demonstrated a scenario in which supply chain process knowledge could be specified and shared amongst business tools [Polyak, 1998f, Polyak *et al.*, 1998]. We participated in many discussions on the National Institute of Standards and Technology's (NIST) Process Specification Language (PSL) [Schlenoff *et al.*, 1996, Knutilla *et al.*, 1998, Schlenoff *et al.*, 1998] project as both active members and Principal Investigator on a connected U.S. Department of Commerce award. This effort was focused on manufacturing process requirements for which we developed a detailed analysis of existing AI planning and scheduling approaches [Polyak and Tate, 1997] and helped to illustrate process representation and interoperability [Polyak, 1997a, Polyak and Aitken, 1998]. Finally, we also played a central role in developing the DARPA/Air Force Research Laboratory (Rome) Planning Initiative's (ARPI) Shared Planning and Activity Representation (SPAR) [Tate, 1998] work products, including a survey of requirements for shared plans [Polyak, 1997b]. All of these efforts share a similar perspective on agents, their purposes and their behaviour.

1.1.3 Research Opportunities

We have begun to point toward some of the open issues, or research opportunities which exist in our intended application of AI planning techniques and representations. For example, we could add the following two:

- How do organisations elicit the requirements for world description knowledge?
- How are shared world and plan descriptions represented and communicated?

An important issue to recognise in applied planning approaches (See 2.3.5) is that it is often the case that many tools, systems, and people participate in the overall lifecycle of the plan or process. It is easy to see that Figure 1.1 only distinguishes simplified design and enactment phases while the overall methodology can be quite complex. Additional phases such as cost-based evaluation and event simulation are typical examples. This implies some need to *integrate* the value provided by each of these optional phases and to incorporate the knowledge obtained. This aspect, combined with our interest in making our approach "common" opens up other issues such as:

- How do we incorporate and support heterogeneous knowledge sources?
- What process-related elements are common to most applications?
- How can we customise and extend the knowledge representation to address specialised needs?
- What type of tools are required to support this approach?

1.2 Research Problem and Hypotheses

We have laid the foundation for an explanation of the problem we are addressing in this thesis. In this section we characterise both the problem and outline the components of our solution. Our approach is based on certain hypotheses that we have tested throughout our thesis work and which we also present here.

The research problem addressed in this thesis can be described from two different viewpoints. We can look at this from an organisational management position or a technology solution position. This roughly corresponds to a top-down and bottom-up perspective. The organisational management or top-down perspective is grounded in organisational requirements and focuses on the problem of

How can we improve the methodology of synthesising and managing organisational process knowledge?

From the technology stance, we are anchored in the knowledge of the capabilities of various tools and approaches and we wish to relate this upwards toward organisational needs

> **How can organisational process design benefit from AI-based planning and plan representations?**

The first perspective helps to guide us toward a relevant, pragmatic solution while the second constrains the possible focus for providing support. In order to understand more of the detail and importance of this problem we will outline a simple scenario. This example is an introduction to the construction scenario which we will go into more depth in Section 6.2.1.

1.2.1 Example: House Building Processes

Consider a scenario in which the organisation is a construction company and the content of the process knowledge is centred around operations for building a house. The business tailors to a set of various building options based on customer needs and requirements. Thus the process of building a particular house is synthesised for and managed throughout a particular building contractual engagement. In order to provide some concrete content for this example, we will draw on a simple construction domain.

A number of planning domains make up a standard library of world descriptions for the O-Plan domain-independent planning system [Currie and Tate, 1991, Tate *et al.*, 1994c, Tate *et al.*, 1996, Tate *et al.*, 1998a] (see Section 2.3.1.4). One of these test domains outlines the fictitious world of the "three little pigs", which is based on the well-known children's story. This domain defines the processes of building houses given the three pig's classic building material selection: straw, sticks or bricks. Material costs and building costs are built into the model. In addition to this there are simple dependencies expressed, such as requiring that the material be purchased before building and that walls must be constructed before adding doors and windows.

From an organisational management perspective we can imagine that we had some conceptualisation of a possible pig house given the details of this domain, e.g. see Figure

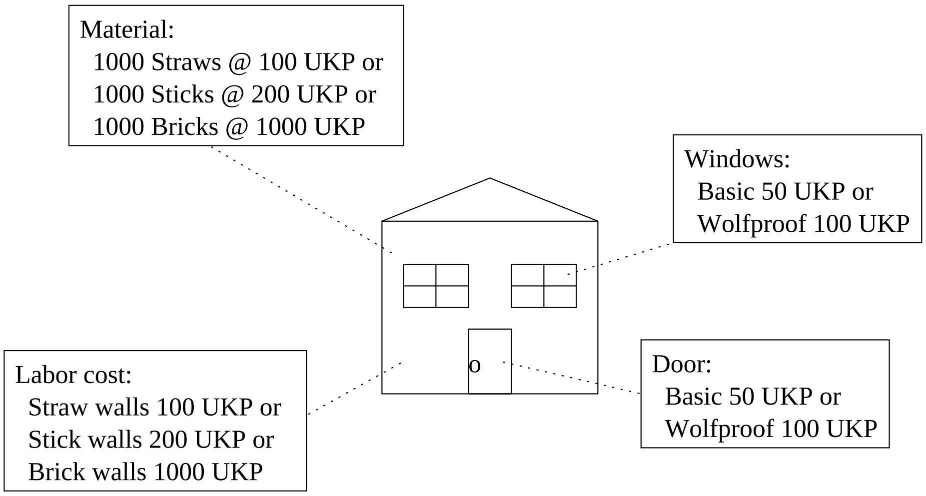

Figure 1.2: Pig house options

1.2. This outlines the building options available to a customer. The conceptualisation in 1.2 could be structured into a set of requirements as we show in Table 1.1.

The company is responsible for outlining methods for synthesising and managing individual house building processes which comply with both the requirements in Table 1.1 and the requirements of a particular order which might state a ceiling spending limit and material preferences. In fact, the process of acquiring and listing these requirements in Table 1.1 is actually part of this overall methodology. It is possible that these requirements might span a range of speciality areas and require knowledge from various subject matter experts, e.g. a brick laying sub-contractor.

During the phase of designing the steps for a construction engagement the company may wish to consider various process configurations which satisfy the constraints. This might require collaboration between various performing and designing agents in which they would need to share knowledge of the unfolding design. Order parameters may need to be temporarily modified to perform "what if" analyses, e.g. if an extra £100 is spent, the client could also have wolfproof windows and doors.

Along with the artifact itself, some knowledge of the rationale might be required to justify individual design decisions. Various tools might be required to support phases such as "construction plan walk-throughs" of completed designs (i.e. simulations) or resource evaluations (e.g. evaluation of the overlap of building material arrival times).

R1.	*A house requires windows, walls, and a door.*
R2.	*Walls must be built from 1000 units of straw, sticks, or bricks.*
R3.	*Wall material must be homogeneous.*
R4.	*Bricks walls are wolfproof.*
R5.	*Windows may either be basic or wolfproof.*
R6.	*Doors may either be basic or wolfproof.*
R7.	*A secure house must have a wolfproof door, wolfproof windows and the walls must be made from wolfproof material.*
R8.	*1 brick costs £1.*
R9.	*1 stick costs 20p.*
R10.	*1 straw costs 10p.*
R11.	*Brick walls incur £1000 for labour.*
R12.	*Stick walls incur £200 for labour.*
R13.	*Straw walls incur £100 for labour.*
R14.	*Labour and parts for a basic door is £50.*
R15.	*Labour and parts for a wolfproof door is £100.*
R16.	*Labour and parts for a basic window set is £50.*
R17.	*Labour and parts for a wolfproof window set is £100.*
R18.	*Wall material must be purchased before walls are constructed.*
R19.	*Walls must be constructed before windows or doors are installed.*

Table 1.1: Pig House Domain Requirements

During enactment at a building site, possible modifications might need to be made to the building process. For example, if straw is unavailable and the contractor is requesting to use sticks instead. The management methodology should be able to help identify which parts of the process are affected and what type of steps need to be taken to rectify the situation or "repair" the process.

In this example, we have outlined a simple scenario which illustrates some of the complexities involved in organisational process management. As we shall see in Chapter 6, these complexities translate to many other scenarios in manufacturing, business, and military environments as well. Effective management of this knowledge represents a significant problem which directly determines an organisation's ability to respond correctly and efficiently.

1.2.2 Hypotheses

Our first set of hypotheses have to do with *representational* issues that we believe are relevant to the problem described in Section 1.2.1.

H1. *Organisational processes can be effectively represented with AI plan descriptions.*

H2. *Knowledge of available actions or potential process options can be effectively represented with AI world descriptions.*

H3. *Process-relatable objects and their relations can be effectively represented within AI world and plan descriptions.*

The first two hypotheses are fairly obvious given the background information which we have provided so far. The third identifies the link between process activities and process objects. This type of link can be found in many process modelling approaches, such as the IDEF3 [Mayer *et al.*, 1992] link between units of behaviour in a process-centred view and object states in an object-centred view. Objects could refer to a number of entities such as the performing agents or required resources (e.g. money and building material given our example). Object relations refers to the fact that these objects may have named connections (e.g. sub-component) or properties (unary relations) (e.g. bricks are wolfproof). The next set of hypothesis have to do with sharing this knowledge.

H4. *Incomplete or completed designs of organisational processes can be shared amongst people or systems.*

H5. *Incomplete or completed designs of world descriptions can be shared amongst people or systems.*

H6. *Rationale of a design can be shared amongst people or systems.*

H7. *The expression of design knowledge can be flexible in order to interoperate with a range of people or systems.*

In this set of hypotheses, we have outlined some of our central expectations for knowledge sharing. The first two (H4,H5) indicate a "workflow" perspective. As in the research on <I-N-OVA> [Tate, 1995, Tate, 1996c], we believe that incomplete designs may include "issues" which represent future action items. These issues appear to connect two distinct spaces: a space of design decisions and a space of organisational behaviour. As we illustrate in Figure 1.3, making a decision on an issue in the decision space leads to a new possible space of behavioral elaboration.

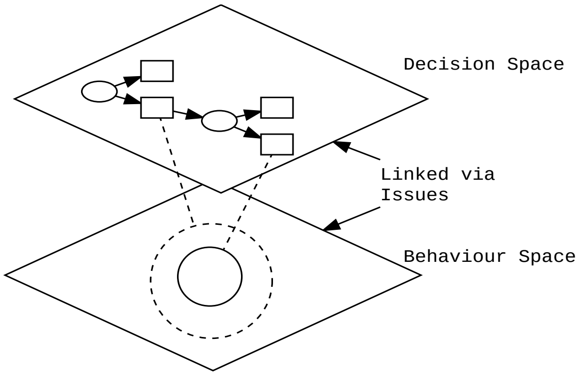

Figure 1.3: Capturing and relating decisions and behaviour. In the upper layer, the ellipses indicate design issues and the boxes indicate alternative options. Moves in the decision space influence the space of behaviour shown on the lower layer.

The rationale for moves in the decision space is an important category of knowledge. We examine this further in our review of research on Design Rationale (DR) (see Section 2.5). Hypothesis H6 represents an expectation which we have for sharing this knowledge alongside the process artifact. More generically we can see that "decision rationale" knowledge is just one example of the specialised information which we may wish to integrate with process knowledge. Another example may be knowledge of a some specific process evaluation. Hypothesis H7 addresses this by stating our view of a flexible representation which can be extended to deal with a range of process related aspects. The final set of hypothesis deal with our expectations of applied tool support.

H8. *A generic tool for visualising and editing organisational processes can be designed which addresses a range of process types.*

H9. *Generic tools for eliciting, visualising and editing world descriptions can be designed which address a range of world types.*

H10. *Efficient system-specific translation may be possible between source and target languages for those systems using shared models.*

We believe that the engineering issues of this approach can be addressed by our work as well. In hypothesis H8 we indicate our belief in a common presentation of process knowledge which applies to many environments (e.g. business or manufacturing). This ideal is similar to those in tools which support other approaches such as Rationale Software's support for UML in Rationale Rose or the Knowledge-Based Systems Inc.'s (KBSI) support for IDEF3 in ProCap. This tool could provide the user with detailed information and valuable feedback on process designs. Similarly, we believe a common approach could be applied to world descriptions as well (H9). In addition to this, we believe tools could be designed to aid in acquiring the required world knowledge. Finally, a challenging hypothesis in H10 is linked to the knowledge sharing hypotheses (H4–H5). We believe that no one language will be heuristically adequate [McCarthy and Hayes, 1969, Wilkins, 1988] for all the tasks involved in managing process knowledge, but it may be possible to translate knowledge to and from a shared language as necessary (see Section 2.7).

1.2.3 Solution Components

Given the hypotheses in Section 1.2.2, we will now outline our solution to the problem we have been describing in this section. This is essentially a prelude to our detailed presentation in Chapter 3 which follows a review of the existing literature.

We describe our approach as building a "Common Process Framework" (CPF). Let's first clarify what we mean by a framework. Wordnet [Millet, 1995] separates the concept of framework into various senses, two of which are relevant here

- The underlying structure or manner of constructing something.
- The structure supporting or containing something.

Our intention is to actually span both of these senses. The CPF is a structure for building organisational processes and world descriptions but is also a structure for containing shared representations of this knowledge. The framework is *common* in that it cuts across industries, but it is also *common* to a range of people or systems which participate in the management lifecycle of this knowledge.

The following set of components comprise the structure of CPF. For each component, we simply provide a short description to convey its general purpose. The details of the implementation of each will be covered in Chapter 3.

1. Common Process Ontology (CPO): We formalised a set of terms and concepts which are considered to be common to most organisational processes. This was based on the <I-N-OVA> constraint model of activity [Tate, 1995, Tate, 1996c] and our experiences in working with process standards projects.

2. Common Process Language (CPL): We developed a sorted, first-order language for expressing world and plan descriptions as design constraints. The lexicon of this language is directly tied to the CPO and is used by all of the tools in the CPF.

3. Common Process Methodology (CPM): We adapted a requirements engineering methodology to structure an approach toward eliciting world description knowledge. This was based on earlier O-Plan research [Wilson, 1984] (see also [Tate *et al.*, 1998b, Tate *et al.*, 1999b]) into utilising the CORE methodology [Mullery, 1979, Curwen, 1991].

4. CPM Toolset: We created a toolset for enacting the CPM and for expressing the requirements work products. The tool supports simple rule-based requirements checking and translation to CPL.

5. Common Domain Editor (CDE): The domain editor addresses visualisation and editing of world descriptions. It can be used to import the initial specification acquired from the CPM toolset and to translate to specific operational languages (e.g. AI planning languages like Task Formalism [Tate *et al.*, 1994a]).

6. Common Process Editor (CPE): A tool for editing individual organisational processes which may have hierarchical structure. CPE utilises a translator which can convert plan descriptions from tools (e.g. AI planners like O-Plan

[Currie and Tate, 1991, Tate *et al.*, 1994c, Tate *et al.*, 1996, Tate *et al.*, 1998a]) into CPL. CPE can also call on export translators which convert CPL processes to specialised target languages.

7. Common Process Assistant (CPA): CPA is an adjunct analysis tool for providing knowledge-based analyses of process knowledge. The functions implemented in this thesis work involved support for reasoning over process temporal relationships.

1.3 Justification for the Research

Given the stated research problem and hypotheses, along with the outline of our solution framework, we next reflect on the justification of this work. The work can be justified by considering a number facts, both pertaining to the problem and our particular approach.

- *Relative neglect of this specific research problem by previous researchers.*

In Erol's Ph.D. thesis on formalising hierarchical planning [Erol, 1995], he noted that the development of planning domains "... is the most neglected aspect of planning, and there is not an established software engineering methodology to guide this job" [Erol, 1995]. Ruth Aylett at the University of Salford concurred: "little work appears to have been carried out as yet either in applying domain modelling techniques specifically to planning problems, or into building general models of planning systems." [Aylett and Jones, 1996]. Steve Chien at JPL posed the question, "Why have so few applications of AI planning been fielded?" [Chien, 1996]. His answer was that it is partially due to a lack of tool support and links to organisational context. Finally, Lee McCluskey at the University of Huddersfield pointed toward a big gap between application/organisation models and AI planning action expansion/goal achievement and suggested the need for a least commitment specification language such as a sorted logic [McCluskey and Porteous, 1997]. While work is beginning on some of this (see Chapter 2), much work remains to be done to address these issues which are connected to our research problem.

- *Importance of this work area.*

This work involves research which attempts to incorporate solutions to problems encountered when transferring AI planning technology to real world scenarios (see Section 1.4). McDermott and Hendler have pointed out that "work on general-purpose planners has primarily occurred at some distance from real problems" and has led to a "split between work on elegant, impractical algorithms and complex, ad hoc, practical programs." [McDermott and Hendler, 1995]. Researchers have called for work to build bridges between theoretically clean research and practical applications of planning [Gil *et al.*, 1995, Aylett and Jones, 1996, McCluskey and Porteous, 1997]. Our approach seeks to build such a bridge and to relate the AI planning model to other fields such as ontological engineering and knowledge sharing in order to show how we can leverage the strengths of various techniques to provide a unified process knowledge management framework.

- *Usefulness of potential applications of the findings.*

While our work on this framework is focused on the development of a specific set of representations, methods, and tools which may be used in future research or applied settings we believe that one of the most useful impacts of the findings could be to encourage integration in the general *practice* of creating and managing organisational processes. This is similar to the stated objectives of the work on CommonKADS [Wielinga *et al.*, 1992, Breuker and van de Velde, 1994] which described the goal as "to improve in some sense an organisational situation through the introduction of a knowledge-based system. The ultimate product of a KBS project is not the KBS but rather the new and improved practice that this system triggers in the organisation."

- *Connecting past and present threads of research*

Finally, we feel that this work is partially justified by the fact that it provides a research link between separate threads of work performed within the O-Plan project. This perspective is explored in more detail in the literature review (see Section 2.13). Our work picks up on earlier efforts with CORE and on the Task Formalism workstation and unites it with the project's current views of issue-based technologies and a constraint model of activity.

1.4 Methodology

In order to ground this work on an applied process framework, we planned out a series of steps to structure our research methodology. The first step involved compiling a set of functional and representational requirements [Polyak, 1997b] for synthesising and managing organisational process knowledge. We present these requirements in Chapter 6 as we use them to illustrate the strengths and limitations of our approach.

In the next step, a subset of these requirements, which are based on the set compiled for NIST's Process Specification Language (PSL), were used to evaluate our selection of <I-N-OVA> versus other major planning and scheduling models as the basis for the common framework language. Our conclusions [Polyak and Tate, 1997] led us to believe that <I-N-OVA> was the best fit for providing the flexibility and comprehensive approach to meet these needs.

In parallel with our identification and development of the framework components, we created a portfolio of organisational process management scenarios which provided representative process examples from construction, business, manufacturing, and military domains. We used these examples both to illustrate and validate our approach and to show how the framework is a common approach to integrating process knowledge. Most of these scenarios were also utilised by other process research projects.

We also established and maintained our focus on the four areas of process management support we cited on page 2: knowledge acquisition, user communication, analysis, and system manipulation. Throughout our presentation of the components in Chapters 3, 4 and 5, we will reference various ways in which these areas were addressed.

1.5 Outline of the Thesis

In this section we provide an overview of the structure of this thesis. We briefly describe the content and purpose of each chapter.

- *Chapter 1: Introduction.* In the introduction we have described the context and motivation for this work. We presented the research problem and our hypotheses. We outlined the components of our approach and a methodology for examining its potential applications. Finally we discussed some justification of the importance and potential benefits of this research.

- *Chapter 2: Literature Review.* In the literature review, we take a detailed tour of the body of related literature in order to show how various aspects of this problem have been addressed in a range of research areas. Our focus is on literature related to domain-independent AI planning, but we fan-out to include research from design and organisational management. Various techniques and representations from the reviewed works helped to build the theoretical foundation upon which this research was based.

- *Chapter 3: Methodology and Design of CPF.* The Common Process Framework is detailed in Chapter 3. We present the methods, representations, phases and architecture which comprise the CPF. We show how the elements of this framework correspond to open research issues which we identified in Chapter 2. This chapter mainly focuses on the constructs required to express a "space of behaviour".

- *Chapter 4: Process Design Space.* Sitting above the "space of behaviour" we can envision a "space of decisions" which we portrayed in Figure 1.3. This space is traversed as we design process artifacts. In this chapter we outline our adaptation of an approach from design rationale to capture and convey the structure of this design space.

- *Chapter 5: CPF Toolset.* In this chapter we present our implementation of the tools required to realize the architecture and phases outlined in Chapter 3. Part of the role of these tools is to unite both the approach towards spaces of behaviour and spaces of decisions into a single point of knowledge management.

- *Chapter 6: Analysis of Scenarios and Requirements.* In Chapter 6 we present a portfolio of organisational process management scenarios which we developed during the period of thesis study. Most of these scenarios were built to meet applied requirements for business, manufacturing and military process representations. For each scenario, we show how the CPF components are used to facilitate the synthesis and management of this knowledge. In addition to this, we present a set of compiled process requirements (both functional and representational) which we use to illustrate both the strengths and limitations of our approach.

- *Chapter 7: Conclusions and Implications.* Finally in Chapter 7 we discuss our conclusions and the implications of our findings. For this purpose we revisit the hypotheses of Chapter 1 and the research issues of Chapter 2. We present the limitations and point toward future work on the framework.

1.6 Delimitations of Scope and Key Assumptions

In this section we outline some of the scope of this work. We explain what is and what is not addressed given the research problem and we show how parts of our scope selection are based on a set of key assumptions.

The idea of improving the methodology of synthesising and managing organisational process knowledge, even anchored by the application of specific technology like domain-independent AI planning and plan representations, still admits a wide range of possibilities. We need to define a notion of research scope which serves to show what this research is and what it is not. We first consider a set of key assumptions we have made about the problem and our approach:

- Organisational process management projects can involve a sizable number of people (expert, specialists, knowledge engineers) who will have to cooperate in an efficient manner.
- Process and world description development can be a very complex task involving numerous significant details.
- The application of AI planning is a very young branch of industry. In order to succeed, it will be necessary to view it as an engineering discipline with industrial techniques.
- Similar to a software system, processes and world descriptions cannot be easily viewed or touched and are much more abstract than corresponding components in other types of engineering (e.g. in the building industry it is usually possible

to "see" how the component parts fit together). In building this knowledge it can be very difficult to understand how the different parts are structured.

Given the first assumption, we have considered the problem from a knowledge sharing perspective. We are interested in the content of what is communicated between these people and the systems they use. We are not directly addressing the mechanism of communication or collaboration beyond the notion that this knowledge may somehow need to be translated to and from various target and source languages. We are concerned with establishing a shared understanding of this knowledge, for which we provide a core set of common concepts.

In the second assumption we acknowledge that in practice, organisational knowledge can involve many complex details. We are not interested in identifying the possible set of all details in any one process type, but rather we are interested in providing a way to extend process representations to customise the content for applicational needs.

The third assumption is on the need for principled techniques or planning "best practice". Most of the work on AI planning has focused on improvements to various ways of generating plans. Our interest is not on improving the model of AI planning, but rather improving the knowledge of how it gets applied. As we have stated, there is much work to be done by the AI planning community to help guide those projects seeking to utilise this work. We are examining ways to build the knowledge for and to utilise the results from a planning system.

Finally, we assume that it is a difficult task for people to visualise and structure organisational knowledge in a way which is analogous to the management of software processes. We are not interested in reinventing the wheel of process modelling, but instead we are focused on adapting techniques to coexist with the AI planning approach.

1.7 Conclusion

This chapter has laid the foundations of this work. In the opening introduction and throughout the presentation of the background information we provided knowledge of the context and motivation of this thesis including an identification of the research opportunities. We discussed the research problem and formed a series of hypotheses about a potential solution. We then proceeded to overview our proposed thesis approach as a set of framework components. We identified justifications of this work and previewed our research methodology. Finally, we pointed out some of our key assumptions which we used to delimit the scope of our research interest. On these foundations, the thesis can proceed with a detailed description of the research.

Chapter 2

Literature Review

At this point the issues related to integrating the synthesis and management of organisational process knowledge have been described and discussed. Our aim is to address these issues with a process framework which is based on AI planning and plan representations. In this chapter we will explore the parent and surrounding disciplines/fields of this research problem, with the aim of charting the body of knowledge in order to show how our approach fits into and relates to it.

2.1 Introduction

We begin our review by establishing an overall model for the research areas covered. The focus will be on the possible application of AI planning and plan representations but we will also widen the scope to consider aspects from fields as diverse as design and management science.

In Section 1.2 we discussed the issues connected to the problem of designing a pragmatic framework for integrating the synthesis and management of organisational process knowledge. On the surface it would appear that this integration could be supported by existing technological innovations and approaches developed within fields such as Artificial Intelligence and Computer Science. In this chapter we provide a review of the work which seems to address parts of this problem and we highlight the research issues which precipitate as we guide you through the various aspects of the body of knowledge. The disciplines/fields involved in this review give rise to various research interests from which we will present important or prototypical examples. The relationships between research fields and areas are outlined in Figure 2.1. As you can see, we will branch out to also consider topics under the general umbrellas of design and management science.

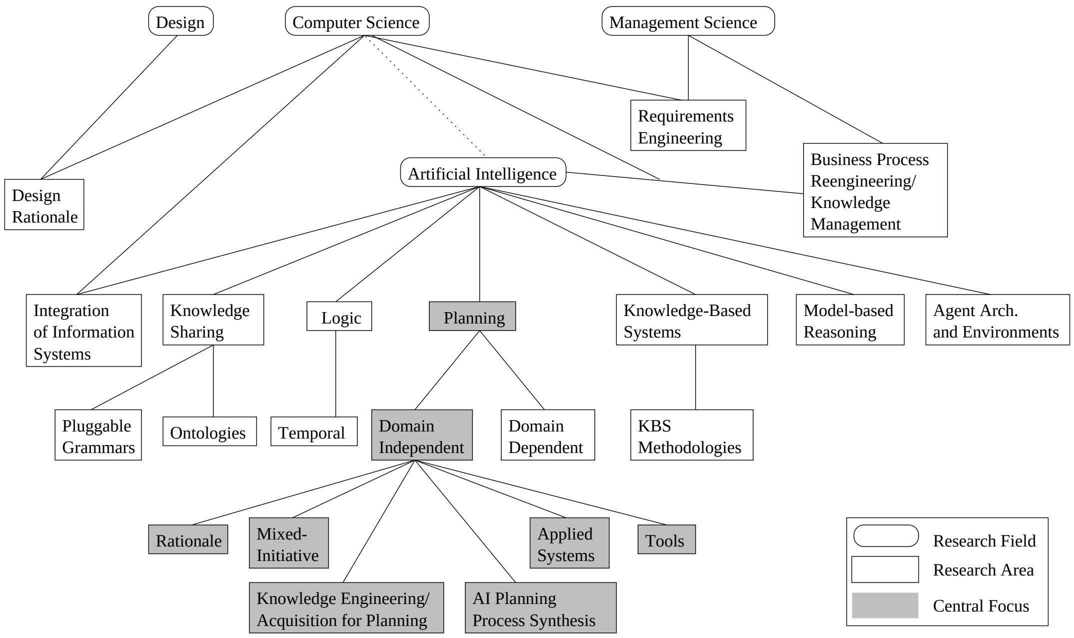

Figure 2.1: Space of fields and areas relevant to this dissertation

2.2 General AI Approaches

The purpose of this section is to show that this review focuses primarily on engineering aspects of Artificial Intelligence. We touch briefly on the symbolic representation of knowledge and introduce early work which led to the development of AI planning.

We begin our review by first addressing our stance on the field of Artificial Intelligence (AI). AI has been defined by a number of researchers in a variety of ways for many purposes. For example, [Charniak and McDermott, 1985] claimed that it is the "study of mental faculties through the use of computational models" with the "ultimate goal of AI research [being] to build a person." They also refer to a view that "AI researchers are trying to create a computer which thinks". More recent texts tend to commit to weaker descriptions such as the statement in [Luger and Stubblefield, 1998], "AI may be defined as a branch of computer science that is concerned with the automation of intelligent behaviour" or in [Rich and Knight, 1991], "AI is the study of how to make computers do things which at the moment people do better". There are many other competing perspectives which we can consider, but it is useful for our purposes to consider a distinction made in [Genesereth and Nilsson, 1988]. This text puts forth the idea that "AI is the study of intelligent behaviour". It then considers this study of behaviour to be composed of two parts: a branch of science and a branch of engineering. The research in each branch tends to be highly interrelated. In this dissertation, we are more closely aligned with the engineering branch of AI in that we are interested in exploring pragmatic solutions to applied scenarios.

One of the important lessons learned by AI researchers has been that "intelligence requires knowledge" [Rich and Knight, 1991]. Over the years a variety of ways of representing that knowledge have been developed (cf. [Brachman and Levesque, 1985]). Most of these approaches to knowledge representation tend to adhere to an underlying assumption about physical symbol systems [Newell and Simon, 1976, Harnad, 1990]. This assumption is embodied by the *physical symbol system hypothesis* which states that "a physical symbol system has the necessary and sufficient means for general intelligent action" [Newell and Simon, 1976]. These symbolic representations of knowledge may be developed to address different perspectives of the world. For example, we may wish to make a distinction between

- *commonsense aspects* of the world (e.g. if an object A is above an object B then it can be inferred that object B is below object A)

- highly *specialised knowledge* for some domain such as with representations of an electrical circuit used in diagnosing faults (cf. [DeKleer and Williams, 1987]). These specialised representations are typically part of "expert systems" designed to address such specific problems in industry.

An AI approach toward the management of organisational processes requires both categories of knowledge. There are certain things we can represent for most processes (e.g. a process has a start and finish point) and other things which may reflect a specialisation for some domain (e.g. a hand soldering process requires solder flux). Some AI researchers have attempted to develop a commonsense representation of processes or plans (cf. [McDermott, 1985], [Davis, 1993, Chapter 9]). One of the most influential pieces of work on commonsense reasoning about processes involved an early attempt at the General Problem Solver (GPS) [Newell and Simon, 1963]. GPS was built to address the problem of performing commonsense reasoning with symbolic manipulations of logical expressions. This research came out of the authors' interest in the psychology of human thinking. In this work, a technique was suggested for directing the search for successful plans by looking at the process activities that directly addressed unsatisfied process goals. This process, labelled means-ends analysis (MEA) connected the presence of some activity with what it achieved and the system searched for a way to incorporate this into an overall course of action. This work helped to spawn one of the oldest and most researched areas of AI: planning.

2.3 AI Planning

In the previous section we introduced one of the major AI precursors to AI planning research, GPS. In this section we characterise what is meant by planning and describe how it relates to scheduling. We introduce the assumptions made in "classical planning" and point out the distinction between domain-dependent and domain-independent planning. Finally we present a chronology of ideas in domain independent planning which we will use as a reference map for looking at the evolution of plan representations.

The early work on GPS [Newell and Simon, 1963] introduced some important terms, concepts, and approaches to understanding human-problem solving which laid the ground work for the branch of AI research known as "planning". The field has grown and changed over time and has acquired a sizeable body of knowledge which addresses various techniques, representations, and methods (cf. [Allen *et al.*, 1990, Weld, 1994, Kambhampati, 1997, Weld, 1999]).

The word "planning" is ubiquitous and its possible interpretation spans many senses. The process of *planning*, in the classic AI sense, can be considered to be "reasoning about the consequences of acting in order to choose from among a set of possible courses of action" [Dean and Kambhampati, 1995]. This synthesised course of action will take an agent from a given initial state to a desired goal state when executed. AI planning research tends to separate out the aspects of this process which are involved with scheduling. *Scheduling* can be considered to be the process of avoiding conflicts of constraints between activities and allocating time and resources to activities within a plan. So while planning is concerned with selecting actions that need to be carried out, scheduling determines when they will be executed and with what resources. In practice, this clear distinction often blurs and real-world problems require interleaving both (cf. [Zweben and Fox, 1994]).

AI planning thus puts forward the idea that an intelligent agent which plans has some set of **goals** which it wishes to achieve. These goals can be met by possibly enacting a course of **action** in an **environment** which the agent may or may not be able to **perceive**. Early research in planning employed some fairly restrictive constraints on this model which collectively came to be known as "classic AI planning". In classic planning, the environment is considered to be static and observable, the actions are deterministic and the agent's perception of the world is perfect. In addition to this, much of the research considered that complete plans could be synthesised prior to plan execution.

Approaches to the classic AI planning scenario and variations of it can be classified into two types: domain-dependent and domain-independent. We can see this division in Figure 2.1. [Tate *et al.*, 1990] characterised these approaches in the following way

- *Domain-Dependent*: uses domain-specific heuristics to control the planner's operation
- *Domain-Independent*: planning knowledge representation and algorithms are expected to work for a reasonably large variety of application domains

Domain-dependent approaches build systems for particular problems whereas domain-independent approaches attempt to capitalise on the shared facets of planning problems so that a problem-solver is only built once. Much of the work on AI planning has been devoted to the domain-independent approach. As we can see from

Figure 2.1 we will be looking at some of the areas that exist under this overall planning category of research.

It is important to note here that ideas on tackling the domain-independent planning approach have evolved over time. These ideas can be chronologically arranged to show the development and separation of the major planning alternatives. Figure 2.2[1] presents these ideas with an emphasis on the major influences and direct descendents. We will use this chronology of ideas as a reference as we examine representations of AI plans. A more detailed chronology which highlights the development of foundational systems and their relation to research areas can be found in [Tate *et al.*, 1990] (pg.27).

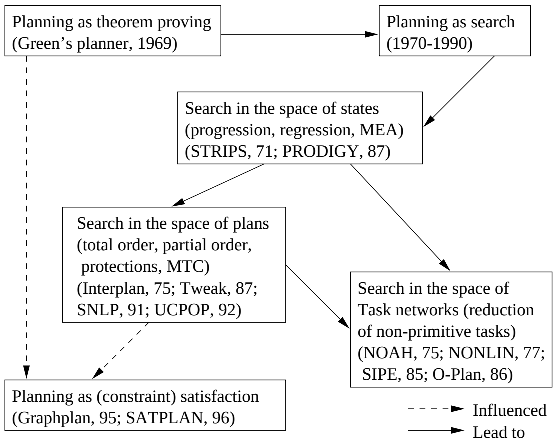

Figure 2.2: Chronology of ideas in domain-independent planning [Kambhampati, 1997]

2.3.1 Plan Representations

In the previous section we introduced a chronology of ideas in domain-independent planning. Our method in this section is to take a structured walk through the progression of these ideas in order to illustrate some of the elements involved in AI plan representations. Our aim is to show how a modern constraint-based view of plan knowledge grew out of this body of research.

As we mentioned, intelligent action (e.g. planning) requires knowledge. AI researchers in planning have developed representations of this knowledge in order to support

[1] This figure is based on a slide from S. Kambhampati's invited talk, "Refinement planning: Status and Prospectus" presented at AAAI-96 in Portland, Oregon. This also appears in [Kambhampati, 1997]

the generation and management of these new courses of action. Figure 2.2 provides us with a reference model from which we can examine some of the aspects of AI plan representations.

2.3.1.1 A Space of States

An early approach to AI planning viewed the process from a theorem proving perspective. As we can see in Figure 2.2 this was realized in 1969 with Green's QA3 program [Green, 1969]. In this work, Green used sets of axioms to represent which actions led to which situations in terms of McCarthy's **situation calculus** [McCarthy and Hayes, 1969]. These axioms were then used to infer action sequences. This approach used logics (see Section 2.4) to represent situations, actions and their effects similar to the operators, states and operator state transformations utilised for search in the GPS work mentioned earlier. This approach led to a number of problems including the frame problem [Hayes, 1973, Shanahan, 1997a]. The frame problem actually involves a few issues (cf. [Georgeff, 1987] pp. 389-391), but the main one most people tend to recognise is the problem of how to determine what changes and what stays unchanged between state transformations.

An alternative to the QA3 theorem proving approach of deducing properties of a situation was offered by the STRIPS problem solver [Fikes and Nilsson, 1971] (see Figure 2.2). The STRIPS approach promoted the idea of editing situation descriptions. Operators, or actions were defined with preconditions and effects (stated as predicate-calculus atomic formulas). The precondition formulas expressed those things that must be true before the action could be applied. The effects were divided into 2 sets: *Add list* - what would be true as a result of applying this action; and *Delete list* - what would NOT be true. The *STRIPS assumption* was that anything not listed in the add or delete lists would not change. This provided a solution to the frame problem[2]. Later work by [Pednault, 1987] united the expressiveness of situation calculus with the STRIPS assumption in the action description language (ADL).

As Figure 2.2 shows, some of this work discussed above enabled searching in the space of states. This meant that in addition to applying the mean-ends analysis (MEA)

[2] The STRIPS approach still underpins many of the current AI planning systems today.

technique developed in the GPS work, AI planning researchers could consider searching for solutions by making refinements in two directions. *Progression* indicates a search initiated from the initial (or possibly some working) state moving towards a goal state by editing the current state description whereas *regression* indicates a search initiated from the goal state moving back towards the initial state (or some desired intermediate state).

2.3.1.2 A Space of Plans

Often, it is easier to see whether or not a given action is relevant to a plan, but much harder to determine the precise position at which a step must occur in the final plan [Kambhampati, 1996b]. Plans manipulated by systems such as the STRIPS problem solver were totally ordered which meant that selecting actions involved making a commitment to a position in a sequence of actions. The upside of this was that we could continue to view planning as a search of states but the downside was that this required both planning and scheduling (see Section 2.3) to be involved at each action selection.

An important divergence from the STRIPS searching method appeared with systems such as HACKER [Sussman, 1973, Sussman, 1974], Interplan [Tate, 1975], NOAH [Sacerdoti, 1975], and Nonlin [Tate, 1977]. For example, Interplan involved a shift away from looking at plans to searching a space of partial plans or "approaches" to partial plans. The Interplan approach still had a "linear" plan development phase though to check the "approach". Later systems maintained partially-ordered plans, i.e. actions could be added to the plan without having to commit to when exactly they would occur. This modification meant though that the plan no longer represented a unique world state and thus characterised a search in a space of partial plans. Some of the contributions of these systems were later clarified by McAllester and Rosenblitt's paper on the SNLP algorithm [McAllester and Rosenblitt, 1991]. One of the most widely known implementations of this algorithm[3], along with an extension for context-dependent effects is UCPOP [Penberthy and Weld, 1992]. UCPOP utilised a significant subset of the ADL language [Pednault, 1987] which was mentioned in Section 2.3.1.1.

A partial plan, as described by the SNLP algorithm, represents a collection of four

[3] These implementations are commonly known as Partial-Order Causal Link (POCL) planners.

things [McDermott and Hendler, 1995]:

- a partially ordered set of steps
- a set of precondition goals associated with each step, which were conditions to be made true before that step in every totally ordered completion of the partial plan
- a set of causal links that commit one step to achieving a precondition of another
- a set of separation links that commit a step to be ordered so that it cannot interfere with a causal link

With this flexibility came increases in plan handling costs. In TWEAK [Chapman, 1987], Chapman describes the modal truth criterion (MTC) which outlines the reasoning involved in determining the truth of statements at any point in the plan. This reasoning is an important part of planning with partially ordered plans and was introduced in the QA (Question and Answering) procedure in Nonlin [Tate, 1977]. Thus, searching a space of plans typically involves a "bookkeeping" strategy. Sussman called the representation of this bookkeeping knowledge the "teleology" of the plan [Sussman, 1973]. The "causal links" referred to in SNLP owe their heritage to Interplan's "Goal Structure" (GOST). The Goal Structure was used to record the link between an effect of one action that was a precondition (or subgoal) of a later one. This knowledge was referred to as validations in PRIAR [Kambhampati, 1989, Kambhampati and Hendler, 1992] and Kambhampati's more recent work uses the term "interval preservation constraint (IPC)" [Kambhampati *et al.*, 1995] to characterise the need to protect the interval defined by this link. It has also been referenced as part of the plan's "rationale" [Wilkins, 1984] which we will return to discuss in Section 2.3.2.

2.3.1.3 Task Networks

As we can see from the SNLP "steps" and "separation links" presented in Section 2.3.1.2, these planning approaches typically represented a plan via its actions and temporal ordering relations. A good example of this was Sacerdoti's procedural nets

[Sacerdoti, 1974]. Action ordering plans express the relationships among actions directly instead of through states and predicates contained within states (as was the case with the situation calculus). The action-oriented approach is preferred for describing complicated causal and temporal relationships between actions in complex domains ([Tate *et al.*, 1990], p.31).

It was also recognised that complex planning domains might benefit from a mechanism that could abstract and relate sets of actions in the domain to make it more manageable. Research in human problem solving suggested an approach to this issue [Pólya, 1945]. This was based on a hierarchical arrangement of knowledge with an increasing amount of detail applied at lower levels of the structure. Hierarchical planners like NOAH [Sacerdoti, 1975] and Nonlin [Tate, 1977] incorporated this idea and introduced another aspect into action ordering plans. The process of searching for sub-reductions of a higher-level action resulted in the inclusion of a plan node or set of nodes as a detailed *expansion* of the plan. The set of expansion nodes could then be attributed to the higher-level node that represented an abstraction of the set.

2.3.1.4 Constraints in Planning

As Joslin pointed out in his Ph.D. thesis work, "virtually any planner that doesn't simply do a brute-force search of the state space can be viewed as doing at least some of its work by posting constraints" [Joslin, 1996]. Thus we can view the information in the GOST (Section 2.3.1.2) or the set of action orderings (Section 2.3.1.3) as causal [Kambhampati, 1994] and temporal constraints [Allen and Koomen, 1983].

The MOLGEN planner [Stefik, 1981] was one of the first AI planning systems which explicitly referred to its knowledge as "constraints". MOLGEN was developed to plan experiments in molecular genetics. It used constraints to represent dependencies between variables that represented objects used in a plan. These constraints once posted helped to guide the selection of actions as interacting plan steps were added. This constraint posting approach is connected to the notion of "least commitment" planning (cf. [Weld, 1994]).

Planning with deadlines and continuous change was tackled in the research on Zeno [Penberthy and Weld, 1994]. This system reasoned with temporal intervals and constraints. This enabled linear programming strategies such as the simplex algorithm to be used for checking for inconsistencies. Collage [Lansky, 1994] is another system which utilised a constraint-based approach. For example, the expansion of abstract activities which we described in Section 2.3.1.3 would be handled by posting a decompose constraint.

Work on planners such as SATPLAN [Kautz and Selman, 1992] and Graphplan [Blum and Furst, 1995] have shown that planning problems can be successfully recast as constraint satisfaction problems (CSPs) [Tsang, 1993]. Also, Joslin and Pollack introduced Descartes [Joslin and Pollack, 1996] which transforms planning problems into dynamic CSPs. Their constraint posting approach was most like MOLGEN but they extended this beyond simple variable binding and made it applicable for all planning decisions.

This strong constraint-based approach was built into the planning architecture of O-Plan [Currie and Tate, 1991]. O-Plan is derived from the earlier Nonlin work and extends its elements such as the GOST and QA procedure which we introduced in Section 2.3.1.2. In addition to this, O-Plan also inherited and improved on the ability to manage complex domain knowledge such as hierarchical network relationships[4] (see Section 2.3.1.3) alongside temporal and resource constraints (cf. [Bell *et al.*, 1986a, Bell *et al.*, 1986b, Vere, 1991, Drabble and Tate, 1994]) and the object variable constraints with we mentioned in the MOLGEN work.

Another important innovation which O-Plan introduced was the incorporation of a blackboard-style [Engelmore and Morgan, 1988] agenda control architecture. Blackboard systems were developed to tackle difficult systems integration issues encountered when different angles of a problem are addressed by separate modules. This made it possible for O-Plan to employ a number of specialised "constraint managers" to work on a plan, all sharing constraints which served to refine the possible course of action (e.g. a resource constraint manager for resource constraints, temporal constraint manager for temporal constraints).

With O-Plan, as with Nonlin, the language used to express knowledge of a plan domain is the Task Formalism (TF) [Tate *et al.*, 1994a]. TF can be used to encode

[4] These artifacts are often referred to as Hierarchical Task Networks (HTNs) in planning research.

various constraints which apply to the domain, domain objects (e.g. resources) or to specific domain actions. A subset of TF, the O-Plan Plan Output Format can be used to express a declarative, frame-like version of a synthesised plan which contains information on the plan actions and temporal relationships. As the O-Plan research matured and moved towards supporting a mixed-initiative approach (see Section 2.3.3) a more general model of the constraints which were underlying the TF operational language was produced. This constraint model of plans is known as <I-N-OVA> [Tate, 1995]. We will return to discuss this work in Section 2.3.4.

2.3.2 Rationale in Planning

In the previous section we traced issues related to the development of AI plan representations and arrived at a work which describes a modern constraint-based view of behaviour. One of the sub-themes in this trace of research, which fed into the development of this model, involves identifying, recording and expressing a plan's rationale. In this section we turn our eye toward this category of knowledge and point out the un-addressed research issues which involve recording plan decision rationale.

Traditional approaches to plan representation focused on the generation of a sequence of actions and orderings. Knowledge rich models, which incorporate plan rationale, provide benefits to the planning process and the use of these plans in a number of ways. In [Polyak and Tate, 1998] we reviewed the use of rationale in AI planning in terms of **causality**, **dependencies**, and **decisions**. We showed how each dimension addresses practical issues in the planning process and adds value to the resultant plan. The contribution of this section is to briefly review this categorisation and to motivate the need to explicitly record and represent rationale knowledge for situated, mixed-initiative planning systems.

Planning rationale can be traced back to the early beginnings of artificial intelligence planning, when the utility of recording such knowledge had been cited [Newell and Simon, 1963, Sussman, 1974, Hayes, 1975]. Rationale has been used in generating plans but has also been applied to other areas of planning as well (e.g. plan analysis, plan execution). "Plan rationale" has been loosely described as "why the

Issue	**Type of Rationale**
Why nodes are in a plan	Causality
Choosing nodes to group into sub-plans	Decisions
Maintenance of truth ranges	Causality, Dependencies
How plan levels connect	Dependencies

Table 2.1: An interpretation of Wilkins' definition of plan rationale

plan is the way it is" [Wilkins, 1988]. Wilkins' more detailed description highlights the multidimensional basis of rationale:

> "The primary tasks of the plan rationale ... are to encode why nodes are in the plan, how nodes should be grouped together into sub-plans that accomplish a goal, how long the truth of a particular goal must be maintained and how different abstraction levels connect" [Wilkins, 1988]

The first item, "why nodes are in a plan", can be viewed as an aspect of **causal** rationale. "How nodes should be grouped" can be considered part of the **decision** rationale of the planning process. The maintenance of truth ranges spans the **dependency** and **causal** rationale while the connection of abstraction levels denotes knowledge in the **dependency** rationale. This interpretation is summarised in Table 2.1.

<u>Causal rationale</u> supports the planning process in a number of powerful ways. In a more general sense, McDermott pointed out: "Causality is fundamental to a lot of problem solving. A problem solver brings things about by causing other things" [McDermott, 1982]. The explicit recording of "what was caused" during planning or causal relationships can be traced back through many of the current and past AI planning systems. This knowledge has been used in:

- controlling search
- connecting plan elements to their purposes
- establishing protection ranges
- ensuring correct planning results

- plan monitoring
- plan interpretation and analysis
- plan execution

In our review of **causality** we looked at the maintenance of causal rationale using the goal structure (GOST) as well as the use of its SNLP counterpart, the causal-link (see Section 2.3.1.2). The representation of a causal-link is a 3-tuple, <s,P,w>, where P is a propositional symbol, w is a step that has P as a prerequisite and s is a step that has P as an effect. This is expressed as: $s \overset{P}{\rightarrow} w$. Causality information is recorded as the result of decisions made by the planning system, but causality can also be explicitly represented in the domain description as well. For example, in SIPE [Wilkins, 1984] and later SIPE-2 [Wilkins, 1988], Wilkins took an innovative approach towards extending the representation of causality by allowing a "causal theory" of a particular domain to be expressed as a set of causal rules, state rules, and init-operators. One of the key contributions of this approach was that actions whose effects are dependent upon world states could be defined without creating specialised operators that correspond to all of the possible situations in which an action takes place [Ludlow and Alguire, 1994].

Additionally in [Polyak and Tate, 1998], we reviewed: Allen's ACAUSE and ECAUSE [Allen, 1984b] which assists in interpreting the causal rationale and interrelationships of plan events and actions; along with another view [Lansky, 1987] which separates causality from eventuality; and Georgeff's [Georgeff, 1987] separation of causality into two types: an event causes the occurrence of a later event, or an event causes the simultaneous occurrence of another event. We also provided more recent examples in which this knowledge has been used in a plan execution agent to detect protection violations while carrying out a plan [Reece and Tate, 1994] and how causality overlaps with conditional aspects [Peot and Smith, 1996] or handling uncertainty [Kushmerick *et al.*, 1995, Dean *et al.*, 1993, Goldman and Boddy, 1994].

Dependency rationale was motivated by early work in planning which pointed out the need to capture such knowledge in plan generation [Hayes, 1975, Stallman and Sussman, 1977, London, 1978]. Dependencies have been used in:

- defining plan element interrelationships

- replanning
- backtracking search
- plan reuse and refitting
- protecting values
- revision of beliefs

As in causal rationale, dependencies can be recorded during planning but in some cases they are computed from the resultant plan network (cf. [Kambhampati, 1989]). Some planners, particularly those that support reuse of previous plans, also save dependencies along with the plan.

Dependencies arise from plan decisions. A planner typically has a number of alternatives to choose from when generating a possible solution to a planning problem. These choices may involve such things as: selecting an operator to achieve a goal, expanding an abstract node, ordering conflicting operators. An option is selected from the possible set, but in order to preserve completeness a planner typically stores the other alternatives as a "choice point". A problem arises when a planner needs to revisit a choice point and select an alternative. Only those aspects which stem from or depend on the invalid alternative should be thrown out. Hayes' solution to this problem came in the form of a "decision graph" which accompanied his journey plan [Hayes, 1975]. A decision graph was used to record the dependencies between planning decisions (dnodes) and nodes in a journey plan (jnodes) as the plan was being built. These dependencies permitted intelligent plan modifications when a new decision needed to be made. Following Hayes' work, decision graphs were also added in 1977 to the Nonlin planner to assist in modifying plans [Daniel, 1983]. Daniel characterised two types of decisions that are made in generating a plan: "choice of expansion for a node", "choice of links to remove an interaction".

In the review, we discussed the link between truth-maintenance systems [Doyle, 1979, de Kleer, 1986] and this early planning work. This gave rise to dependency-directed backtracking, which permitted the maintenance of a planning system's nonmonotonic belief set (cf. [Ginsberg, 1993, Kambhampati, 1996a]). Work continues to be done on incorporating TMS or "reason maintenance systems" into planning [Doyle, 1994, Doyle, 1996]. The current focus is on an "incremental application" that is flexible and customisable to the planning purposes.

Two of the most important efforts involved with dependency rationale and plan reuse include: PRIAR [Kambhampati, 1989, Kambhampati and Hendler, 1992] and Prodigy/Analogy [Veloso, 1996]. The PRIAR system annotated plans with information about the dependency structure between operators. These annotations were comprised of sets of "validations". A single validation in PRIAR was a 4-tuple $\langle E, n_s, C, n_d \rangle$ where n_s and n_d are leaf nodes belonging to the hierarchical task network (HTN) and E is the supporting effect of n_s used to satisfy the applicability condition, C, of node n_d. One set would contain the validations that were supplied to other nodes, another set would contain validations that were consumed by the node, and a final set would contain validations that were required to "hold" over the node. In Prodigy/Analogy dependency information is recorded in a plan's justification structure. Nodes in this structure are incrementally added at decision points. These nodes contain slots of recorded plan data. One of the three main kinds of justifications used captures links among choices in the subgoaling structure. Dependency annotations from slots like, "precond-of" and "relevant-to" are saved along with a successful solution to the problem.

Dependencies are often governed by constraints in the domain so we also examined domain dependency representations as well. For example, ADL (Section 2.3.1.1) addressed the need for a more expressive language that could tackle the dynamic nature of continuous processes and simultaneous actions. ADL has an advantage over other plan representations in that explicit dependencies between circumstances and an action's effects can be efficiently expressed. The dependency relationships are encoded into the domain itself between a plan operator and dynamic aspects of the domain, rather than only being expressed statically between two operators. We also reviewed the use of Nonlin and O-Plan's condition types (e.g. supervised, unsupervised) for establishing domain dependencies between operators (cf. [Tate *et al.*, 1994b]).

Decision rationale underlies the expression of planning causality and dependency relationships. In pragmatic organisational contexts, people and machines decide together how to solve tasks, formulate a domain or plan, execute a plan and so on. Recording the rationale of these decisions adds value to the planning process in the following ways:

- facilitation of communication and reasoning
- promoting a shared understanding of beliefs and intentions
- maintaining a consistent approach
- connecting agents to their responsibility in the plan process
- helping to steer the decision-making process

This axis represents the area of plan rationale which has received the least amount of attention in the planning research literature. Two important exceptions to this include the work on TRAINS [Ferguson and Allen, 1994] and Prodigy [Veloso *et al.*, 1998].

Ferguson and Allen constructed a formal model of plans based on defeasible argument systems in their TRAINS project [Ferguson and Allen, 1994]. This model allows for an explicit representation of plans as arguments that a course of action under certain explicit conditions will achieve certain explicit goals. This certainty is achieved by developing defeasible arguments which are sets of argument steps that can play roles like: rebuts, conflicts, undercuts. The overall argument then can be said to be defeated or undefeated. The incremental nature of argumentation lends itself to the construction of reasoners that have to understand the reasons of other agents and communicate with them. The Prodigy research [Carbonell *et al.*, 1990, Veloso *et al.*, 1995] has also been moving toward a decision rationale perspective [Veloso, 1996]. For example, the user of this system can exercise complete control over the planning decisions. This control allows manual direction of the developing plan and annotation of nodes with rationale. Involved human agents can attach "guidance" to the plan as justification for the plan structure. Veloso et al.'s more recent work has focused on synthesising rationale-based monitors which can detect changes in decision criteria that may require plan changes in dynamic environments [Veloso *et al.*, 1998].

Work in design rationale (DR) [Moran and Carroll, 1996] suggests one method of tackling the new forays into representing and communicating planning decision rationale. Possible research into applying DR techniques to planning could help define new ontological elements required to represent plan knowledge. We will address this link in Section 2.5. Some of these issues in decision rationale have also been considered in requirements engineering (RE) [Davis, 1993, Sommerville and Sawyer, 1997]. Essentially, design specifications and requirements serve to constrain the possible space of a software system implementation in much the same way that a plan can be constrained. We will look into this in Section 2.11.

In conclusion, it appears that rationale has been a key component in the planning process and will continue to increase in importance. It has been used to improve the way a planner reasons about a plan and manages the details of plan element relationships. The three dimensions reviewed are strongly interrelated and highlight a multidimensional contribution. Some aspects, especially causal and dependency rationale, can be traced back to early work in planning. Planning decision rationale is beginning to gain more attention as deeper levels of organisational integration are required. Plan representations are expected to mature and evolve, at least in part, by incorporating rationale which can support collaboration between human and machine-based planners. This type of exchange has been termed "mixed-initiative" [Burstein and McDermott, 1994] and is the subject of our review in Section 2.3.3.

2.3.3 Mixed-Initiative Planning

In Section 2.3.1 we looked at some of the issues in plan representation as ideas in AI planning evolved over time. This review led us to recent research on representing plans as a set of constraints. This constraint-based representation has been described as being appropriate for supporting mixed-initiative planning. Additionally, in Section 2.3.2 we examined the role of rationale knowledge which indicated that more work on planning decision rationale is required to support mixed-initiative planning. In this section we describe mixed-initiative planning. We consider some of the research issues in this area which relate both to the constraint-based representation and the need to understand the decisions behind it.

From the planning literature, we can see that planning systems that are situated in an organisation typically need to work in cooperation with a variety of agents. This may mean that humans and machines collaborate in the development and management of plans while sharing a common initiative. This has been termed "mixed-initiative planning" [Burstein and McDermott, 1994]. With a large number of people and systems working together to produce a solution, agents may need to communicate intentions, beliefs, and justifications. When a decision is to be made, machine or human, the ramifications need to be considered within a "shared understanding".

Consider a simple model in which two human beings are cooperating in the creation of a plan. What is important knowledge for them to share? Gross et al. conducted a study in which two planners communicated via a microphone to collaborate on plan formation [Gross *et al.*, 1993]. In no case did the planners simply convey the plan as a set of actions. The agents identified goals and sub-goals, identified important actions, stated relevant facts that would help in the development of the plan, identified problems with what the other agent proposed, requested clarification, confirmed each others suggestions. Another study came to the same result with only a relatively small percentage of the discussion concerned with adding or refining actions [Allen *et al.*, 1996]. This suggests that a richer model of plans is necessary to convey key pieces of knowledge needed to make planning decisions when human beings are involved.

An example of an interactive planning architecture which supports this mixed-initiative, decision-making approach is Perini and Ricci's forest fire fighting system [Perini and Ricci, 1996]. They illustrate that "in some cases [a human agent] is able to solve the current goal, for example mostly regarding strategical decisions, in other cases he wants to set constraints on the search process". Constraints may be placed by humans or a planning system, e.g. in modifying duration of an action, changing begin or end times, removing/adding actions. Similar work has been described in a search and rescue domain for the Rescue Co-ordination Centre (RCC) at Pitreavie near Edinburgh [Cottam *et al.*, 1995]. This involved system support for the allocation, application, and co-ordination of military assets for search and rescue planning.

Another crisis-solving planner which is based on constraint-oriented management is DIPART [Pollack, 1996]. DIPART consists of a network of communicating nodes each assisting a human planner along with a simulated environment which introduces crisis events. One of the main contributions of this approach is the description of how control and communication during planning is managed in this dynamic environment.

This trend toward integrating a range of AI systems and the inputs from human users has been a central vision of the ARPA-Rome Laboratory Knowledge-Based Planning and Scheduling Initiative (ARPI) since its very inception [Fowler *et al.*, 1996, Tate, 1996a]. Part of this vision outlined a distributed collaborative mixed-initiative planning process (cf. [Wilkins and Desimone, 1994]). The interaction described in this vision was exemplified in the joint ARPI work between the TRAINS and O-Plan projects which was described by Tate [Tate, 1997]. This collaboration discussed the possibility of blending the multi-modal user dialog capabilities of TRAINS [Allen *et al.*, 1996] with the flexible, modular O-Plan planning system [Currie and Tate, 1991]. A richer interface was anticipated between these two agents to support this mixed-initiative planning environment.

As further evidence of the growth in mixed-initiative approaches, it should be noted that the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98) held a separate workshop to present papers and discuss issues related to interactive and collaborative planning. The workshop was motivated by a need to integrate automated systems with the abilities of human planners and decision makers. This workshop focused on having the human "in the loop" as both a practical necessity and an intellectual opportunity.

One of the central aspects in mixed-initiative planning revolves around an ability to communicate plans and plan related knowledge which we will examine in Section 2.3.4.

2.3.4 Shared Plan/Process Projects

Up to this point we have looked at the development of AI plan representations and delved into planning rationale knowledge. Given this backdrop, we discussed mixed-initiative planning and pointed toward an important aspect which needs to be addressed in this approach. This aspect involves the communication of plans and plan-related knowledge. In this section we overview some of the major projects involved with providing a shared set of terms and concepts for planning or process information.

"What is the point of forming plans?" This question was explored in [Pollack, 1992] and was shown to be the motivation behind the work on the Intelligent, Resource-Bounded Machine Architecture (IRMA) [Bratman *et al.*, 1988]. This discussion, as well as one aspect of IRMA, was linked to Bratman's fundamental insight that "agents commit to their plans, which then frame, and thereby constrain, their subsequent reasoning" [Bratman, 1987]. In effect, it is typically not the case that a plan is simply designed to be executed as it was with early planning work (cf. Shakey the robot [Nilsson, 1984]), but rather a plan may be used to tell other agents what to reason about and what not to reason about.

This observation is a very important one from the standpoint of a mixed-initiative approach (Section 2.3.3). In participating in a planning process, agents expect to receive some knowledge of the plan which informs them of the current state of planning affairs (e.g. which actions are required, what orderings have been imposed, which goals or constraints remain unaddressed). This knowledge then drives their reasoning which may involve further planning or replanning, plan evaluation and assessment, or other specialised applications such as resource scheduling and load balancing along with plan execution.

The heart of this matter involves knowledge sharing which we will return to discuss in Section 2.7. For the purpose of this section though we will make a distinction between those efforts involved with providing a *shared interface* and those providing a *shared representation*. These aspects are related and were outlined in the ARPA knowledge-sharing effort [Neches *et al.*, 1991].

2.3.4.1 Shared Interfaces

Some efforts have focused on the provision of an interface by which agents can at runtime, or in our case during the planning process, query other agents or a knowledge source to obtain information (e.g. knowledge about a plan or process). The interface advocated in [Neches *et al.*, 1991] is the Knowledge Query and Manipulation Language (KQML) [Finin, 1992]. An alternative offered by the work on the Generic Frame Protocol (GFP) [Karp *et al.*, 1995] provides an interface which is "grounded in knowledge representation structures" as opposed to KQML's performatives for agent execution. In addition to these, we may add O-Plan's more system-specific interfaces for plugging in various constraint managers, knowledge sources, plan world viewers, etc. [Currie and Tate, 1991, Tate, 1994a].

2.3.4.2 Shared Representations

A complement to research on a shared interface involves research on the content of what is being shared. As we said earlier, we will return to the more generic topic of shared representations in Section 2.7 but in the remainder of this section we will reference work on shared plans and processes.

There have been a number of initiatives to standardise shared languages, or "interlinguas", within the general subject area of activities and processes. These efforts span a range of applications and environments including: enterprise processes in PIF (the Process Interchange Format [Lee *et al.*, 1996, Lee *et al.*, 1998b, Lee *et al.*, 1998a]); workflow processes (International Workflow Management Coalition [WfMC, 1994]); and manufacturing processes (NIST's Process Specification Language [Schlenoff *et al.*, 1996]). Work on both PIF and PSL have established an approach in which a core set of concepts are defined (via logical axioms) and which may be extended through the use of partially shared views (PSVs) [Malone and Lee, 1990].

All three of the efforts cited above have benefited from past and present work on AI-planning based knowledge sharing work. Probably the first major undertaking of this effort was under the DARPA/Air Force Research Laboratory (Rome) Planning Initiative (ARPI) [Fowler *et al.*, 1996] in which a number of participants created the KRSL plan language [Lehrer, 1993]. KRSL outlined a list of concepts and relations for expressing shared plan knowledge (e.g. plan, event, time-interval, duration, objects, resources, etc.). KRSL did not receive much acceptance in the planning community though and it has been criticised as having too rigid of a structure as well as excluding much that was already being done within AI planning research [Tate, 1998].

A subsequent effort looked into the possibility of revisiting this work in order to extract a smaller core plan *ontology* (See Section 2.7.1). This effort, called KRSL-Plans, was unfortunately not brought to a conclusion though it did lead to an outline plan model [Tate, 1996b]. The most recent effort involved in this vein of research is the work on the Shared Planning and Activity Representation (SPAR) [Tate, 1998]. The working group of this effort produced a draft of the SPAR approach and a planning model based on the earlier inputs (e.g. KRSL, and KSL-Plans) along with knowledge gained from other planning/process sharing research work.

2.3.4.3 <I-N-OVA>

One of the most significant inputs to this work was the <I-N-OVA> constraint model of activity which we introduced in Section 2.3.1.4. <I-N-OVA> [Tate, 1995, Tate, 1996c] is a common representation of tasks, plans, processes and activities based on the notion that these are all constraints on behaviour. The name suggests that a plan can be thought of as a tuple of such constraints where "I" represents a set of outstanding issues (e.g. pending constraints), "N" represents an anchoring set of node constraints (e.g. to include or not include nodes), "O" and "V" represent critical constraints related to orderings and variables, and the remaining possible constraints are grouped under the "A" set for auxiliary. Thus, "Planning is the taking of planning decisions (I) which selects the activities to perform (N) which creates, modifies or uses the plan objects or products (V) in the correct time (O) within the authority, resources and other constraints specified (A)" [Tate, 1995]. An informal plan ontology based on this model was also outlined in [Tate, 1996d].

2.3.4.4 More Shared Representations

A number of other projects have also or are currently tackling/tackled the issue of developing a shared representation of processes and plans. For example, the efforts of the Object Model Working Group (OMWG) are focused on the development of an object-oriented representation called the Core Plan Representation (CPR) [Pease and Carrico, 1999] which also drew its inputs from several of the works cited above and is intended as an object-oriented refined version of SPAR.

OZONE [Smith and Becker, 1997] is another example which entails a toolkit for configuring constraint-based scheduling systems[5]. A central component of OZONE is its scheduling ontology, which defines a reusable and extensible base of concepts for

[5] This builds on earlier work with OPIS [Smith, 1994]

describing and representing scheduling problems, domains and constraints. OZONE adopts an activity-centred modelling viewpoint. There are five basic concepts of the ontology - Demand, Activity, Resource, Product, and Constraint. The ontology also defines specific inter-relationships and properties for these entities.

Traditionally, plan generation and reactive execution have been considered as separate activities, with few attempts to integrate them within a single system. The Act formalism [Wilkins and Myers, 1995] is a language for representing the knowledge required to support both the generation of complex plans and reactive execution of those plans in dynamic environments. Act has been used as the interlingua in an implemented system that links a planner (SIPE-2 [Wilkins, 1984, Wilkins, 1988]) with an executor (PRS [Georgeff *et al.*, 1989, Georgeff and Ingrand, 1989]). Act is intended to serve as a general-purpose representation language that could be used to share knowledge between many different execution and planning systems. The representational and computational adequacy of Act has been validated by implementing the Cypress system [Wilkins and Myers, 1995], which uses Act as an interlingua to enable runtime interactions between planning and execution subsystems.

In many ways, the Planning Domain Definition Language (PDDL)[6] which was developed for the AIPS-98 planning competition [Simmons *et al.*, 1998] can also be considered to be a shared language. The PDDL's expressiveness is roughly equivalent to ADL (see Section 2.3.1.1) and it provided a mechanism by which a set of "requirement" flags could be included in a representation to indicate which kinds of conceptual extensions were required (e.g. conditional effects, disjunctive preconditions, etc.)

Part 49 (Process structure and properties) [ISO, 1995] is an integrated generic resource of STEP (Standard for the Exchange of Product model data) written in EXPRESS. It specifies the information necessary to specify the actions or potential actions to realize a process. This includes the relationships between the actions or potential actions in the process and the relationships between the processes that are used to realize a product. A process plan is the specification of instructions to realize a product. This part does not specify any particular process, but defines the elements to exchange process information. This part is applicable to all types of process definitions that can

[6] The PDDL is also known as Classical Planning Problem Specification Language (CPPSL).

be represented in a discrete manner.

Some of these shared representations reflect greater insight into pragmatic knowledge sharing issues as a direct result of experience gained in applying these ideas to various real-world problems. In the following section we will provide some examples of this applied work.

2.3.5 Applied Planning Systems

In Section 2.3.3 we introduced mixed-initiative planning and discussed some of the approaches to supporting this planning model. We pointed toward one of the underlying challenges of this approach which involves effective sharing of plan/process knowledge. We examined projects which have attempted to address this challenge in Section 2.3.4. Some of these projects have benefited from experienced gained from applied planning systems. In this section we briefly outline some of the major applied planning systems and consider some of the lessons learned.

An increasing number of requirements are placed on both AI planning systems and plan representations in a move towards applied settings. In many ways, the majority of planners have been scaled to work on small to medium problems and their plan representations were tailored for specific use by planning systems. For example, Khambhampati talks about the issues in scaling-up refinement planners [Kambhampati, 1997] in which he notes that most of the existing planners scale up poorly when presented with large problems. Many of the scenarios addressed have been simplified, research-based applications. A number of exceptions to this are planning systems that have been implemented in practical, real world situations.

Langley and Drummond state, "for engineering development and technology transfer purposes, tasks that include practical difficulties will be more useful [than artificial domains]" [Langley and Drummond, 1990]. Practical planners require a "knowledge rich" model that allows them to integrate efficiently given the demands of the surrounding environment. This viewpoint was underscored in [Valente, 1995]

"The more powerful, richer and more adequate to the problem the world

> representation is, the more likely that it is that the planner operates adequately in the specific application domain."

An example of this type of applied planner is Optimum-AIV [Aarup *et al.*, 1994, Arentoft *et al.*, 1991]. Optimum-AIV is a planner implemented at the European Space Agency that is used in the assembly, integration, and verification (AIV) of spacecraft. This planner is accessible to managers that require a detailed level of interaction and control over plans. One example of the "richness" of Optimum-AIV's representation is in the recording of planning decisions (see Section 2.3.2) to explain the rationale of the plan. Optimum-AIV is based on the open planning architecture defined in the O-Plan research [Currie and Tate, 1991]. The O-Plan planner and its techniques have also been used in a number of challenging environments including: back axle assembly process planning at Jaguar Cars, software procurement planning at Price Waterhouse, mission planning for the ERS-1 spacecraft [Fuchs *et al.*, 1990] and factory production planning at Hitachi. TOSCA is another example of a system based on O-Plan [Beck, 1993]. TOSCA has been implemented at Hitachi for job shop planning and scheduling and contains a rich representation of the capacity and setup constraints and objectives.

A number of other applied planning systems that rely on richer representations could be added to this set as well. For example, SIPE-2 has been used to plan emergency responses to oil spills [Agosta, 1994] and to integrate planning for military air campaigns [Wilkins and Desimone, 1994]. AI planning has also been used in: mission scheduling for spacecrafts [Drabble, 1990], automatically generating procedures for processing space image data [Chien and Mortenson, 1996], decision support for controlling deep space network antenna operations [Chien *et al.*, 1996] and scheduling the hubble space telescope [Johnston and Adorf, 1992]. A collection of some of these applied, real world systems are overviewed in [Knoblock(ed.), 1996].

Some planning researchers have pointed out that applied knowledge-based plan representations developed for AI planning can be employed for many uses beyond generative AI planning [Tate, 1994b, Bratman, 1987]. The term "knowledge-rich" was first applied to plan representations that were used in the Interactive Planner's Assistant (IPA) developed by the UK Alvey Programme's PLANIT Community Club in 1986 [Drummond and Tate, 1992]. The primary contribution of this research was not in plan generation, but rather the use of representations in improving the monitoring, analysis and advisory capabilities. In this research, plan representations were used to augment project planning, process planning, and job shop centre scheduling. Plan representations have also been suggested as a source of support for business process reengineering, process automation, process modelling, and workflow management [Tate, 1994b].

As these implementations show, representations are now required which weave together specialised knowledge and progress on a variety of topics, techniques, and standards involved in complex domains. A number of planning researchers have pointed out the need to bridge theoretically clean research and practical applications of planning (cf, [McDermott and Hendler, 1995, Gil *et al.*, 1995, McCluskey and Porteous, 1997, Jarvis and Winstanley, 1998]). Chien considered this point when he asked, "Why have so few applications of AI planning been fielded?". The answer, as we said earlier, is that he believes it's partially due to lack of tool support and links to organisational context [Chien, 1996]. In the next section, we will examine some of the literature on existing planning tools which help to integrate AI planning into an organisation.

2.3.6 Planning Tools

> *Up to this point in the literature review, we have examined a number of aspects of AI planning. We discussed plan representations and planning rationale knowledge and considered mixed-initiative and applied planning along with various approaches to shared representations. At the end of Section 2.3.5 we highlighted an important research issue which involves the provision of adequate tool support for domain independent planning. In this section we review some of the existing tools which have been reported in the literature and outline their contributions.*

What types of tools are required to make effective use of domain independent AI planning systems and plan representations? With a few notable exceptions, it appears that AI planning researchers are only beginning to address this question. Tools are required for a range of reasons which span the lifecycle of domain and plan knowledge within an organisation.

Some researchers have attempted a planning tool box approach, for example qwertz, which entails a set of software modules that could be combined to build different planners (generic and/or application) allowing users to add and use their own modules [Gordon *et al.*, 1993b, Hertzberg, 1996]. While this particular effort fell short of its intended goals, it did produce some interesting insights and lessons for planning research (cf. [Gordon *et al.*, 1993a, Thiébaux, 1995]). One of the impediments to the qwertz work was summed up in the following statement [Hertzberg, 1996]: "While there is a large corpus of literature on the planning process and its details, there is not enough work about

- *Knowledge acquisition for planning*: How to get domain knowledge into a planner?
- *User interfaces for planners*: How to represent the planning results in a way that human users can easily understand and handle?"

This statement outlines two categories of planning tool research to consider. We will briefly review some of the work that has been reported on each. We acknowledge that the notion of "planning tools" could be expanded to refer to a number of other categories which might correspond to specialised reasoning modules (e.g. constraint managers, scheduling modules, qwertz's software modules) but we will focus our discussion here on the cited tool areas.

2.3.6.1 KA Tools for Planning

This area has begun to grow over recent years and we will review some of the major advances in planning knowledge acquisition and engineering in Section 2.3.7. As a prelude to this review though, we will first consider some of the tool-based issues and lessons learned from an applied planning system (see Section 2.3.5), JPL's multimission VICAR planner [Chien and Mortenson, 1996, Chien, 1996].

In [Chien, 1996], the researchers concluded that at least some of the tools needed to support planning knowledge acquisition (KA) are:

- tools to allow domain experts to create and debug their own planning knowledge bases

- tools for software verification, validation and testing
- tools to facilitate updates and maintenance of the planning knowledge base

In this work, they showed that many types of knowledge encoding errors can occur (e.g. incorrectly defined preconditions, incorrectly defined effects, incorrect variable specifications). The ramification of these errors often produced one of the following end symptoms: incorrect plan generation or failure to generate plan. While the former can be addressed by using the plan to debug a fault in the domain knowledge, the latter is far more difficult. This led to the implementation of two types of tools which are characterised by the following techniques

- *static KB analysis techniques* to detect certain classes of syntactic errors in a planning knowledge base
- *completion analysis techniques* to interactively debug the planning knowledge base

This type of domain checking was also advocated in earlier research which suggested that a requirements engineering methodology could be adapted to structure such kinds of analyses [Wilson, 1984]. The Controlled Requirements Expression (CORE) [Mullery, 1979, Curwen, 1991] was proposed for structuring these domain management activities. The tool-based support was to be provided via a hook for an expert system-style agent interface to the Task Formalism workstation [Tate and Currie, 1984, Tate and Currie, 1985] (see Section 2.3.6.2) which would provide various services such as searching for close matches for terminological differences or incomplete information. This also included some standard checking based on CORE analysis techniques:

- Does every activity node have at least one precursor and one successor?
- For every node which has a precondition, is the precondition satisfied by the current network or by another node at the same level or higher?
- Do precursor and successor assignments match?

Unfortunately, this research was set aside once the initial prototype was completed. We will return to consider CORE and its possible application to engineering planning knowledge in Section 2.11.

2.3.6.2 User Interfaces for Planning

While we motivated this research area with a question related to human manageable representations of *planning results*, we can expand this scope to consider human manageable representations of *planning domains* as well. This in fact intersects with the research area described in Section 2.3.6.1. The distinction is then blurred between the two areas, but our focus here will be more on the presentation, visualisation, and editing of planning knowledge as opposed to the underlying acquisition or engineering techniques discussed in Section 2.3.7.

Many of the major AI planning systems have developed some kind of interface in order to support aspects of the planning process. Some tools have been designed to link into an openly controllable planning architecture which help visualise and even alter the planning process. For example, the Prodigy system [Carbonell *et al.*, 1990, Veloso *et al.*, 1995] has an interface for "running a planning domain" in addition to supporting the building of the domain [Blythe *et al.*, 1996]. A human user can step through and interrupt the planning process as well as provide choices for the planning decisions. This is visualised with a node-arc graph along with a set of menus presenting various choices and user responses.

One of the tools used to support the SIPE-2 planner is the Act-Editor [Wilkins *et al.*, 1995, Myers and Wilkins, 1997] (see Section 2.3.4.2 on Acts). This editor also supports a node-arc presentation for displaying, editing, and inputting acts. Nodes may be of various types such as goal, primitive action, conditions which are already true, or split/join nodes. The tool includes helpful utilities such as a simplifier which streamlines the structure of an Act eliminating unnecessary plot nodes and redundant ordering links [Wilkins and Myers, 1995]. A similar tool for managing plan operators and which is used while planning (e.g. in the SOCAP planning system) is desJardins' operator editor [desJardins, 1996].

Other tools have been developed which simply record the actions taken by the

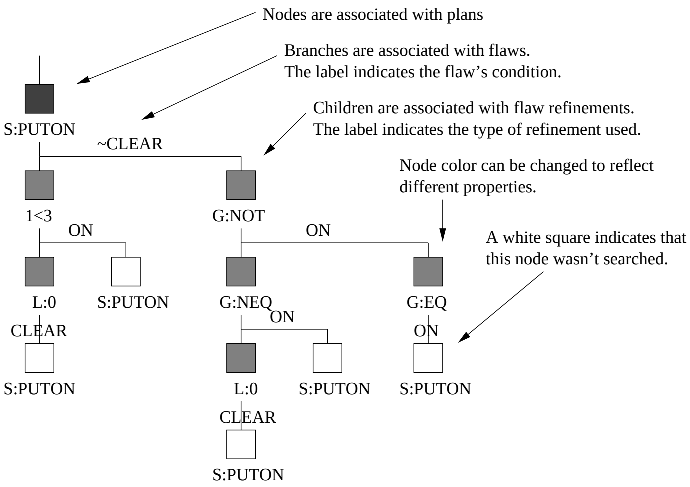

Figure 2.3: Visualising UCPOP plans in PDB

planner and then reconstruct that information for the user in a meaningful way. For example, the work on the graphical plan debugger (PDB) [Kwok, 1995] focused on recording and presenting plan-space search trees generated by UCPOP [Penberthy and Weld, 1992]. This is illustrated in Figure 2.3. In the graphic plan space display, the left hand node is always the selected option. The label below a node refers to the reason why it is in the plan. For example, "S:PUTON" means a new step "PUTON" has been added, "L:0" means a new link to step 0 has been added and "1<3" meant that an ordering constraint (promotion or demotion) had been added. This type of tool could be used to help debug incorrect plans as suggested in Section 2.3.6.1.

The current version of O-Plan [Currie and Tate, 1991] offers support for controlling the planning process via its "developers menu" which allows a user to break in and inspect the planning state. All of the information about the plan, e.g. nodes, variables, and teleology (see Section 2.3.1.2) can be textually output to a window. O-Plan has a limited capacity for graphically visualising completed plans via a postscript output but its more powerful interface lies in the link to external plan viewers [Tate and Drabble, 1995].

Unfortunately, O-Plan *currently* provides nothing like the Act-Editor for managing plan domain knowledge which is expressed in its native Task Formalism [Tate *et al.*, 1994a]. As we pointed out in Section 2.3.6.1 though, there was early work on the Task Formalism Workstation [Tate and Currie, 1984, Tate and Currie, 1985][7]. In fact, domain capture and modelling has been an issue in Edinburgh-based planning research as early as the work on the Nonlin [Tate, 1977] planner. The original O-Plan overall architecture and system design, which dates from 1983, outlined a need for a defined methodology and toolset which would guide users performing various roles in the acquisition and analysis of domain requirements for planning [Currie and Tate, 1991]. Early prototyping efforts on the Three Rivers PERQ-based TF Workstation demonstrated tool-support for the domain modellers (an expert providing the structure of the domain and specialists providing the details) and planners (acting in any one of a range of roles).

One of the trends which is emerging in AI planning involves multiple presentations of planning knowledge which are specialised for particular environments[8]. For example, a manufacturer might prefer to look at a synthesised process plan from a material flow perspective expressed via a State Task Network (STN) [Kondili *et al.*, 1993]. This point was covered in [Drabble, 1995] which stated that each system involved in planning has its own perspective on the planning problem and must be capable of communicating in a way that allows other systems to assimilate new information into their perspective of the problem. Drabble advocated, "an intelligent planning tool [which] stores everything it learns in an adaptable form so it can inform the user of which solutions from other areas can fit together to solve the current tasks and needs."

The NIST PSL project [Schlenoff *et al.*, 1996], which we introduced in Section 2.3.4.2, had built the concept of multiple presentations into its approach. The project split its efforts into three parts: semantics, scenarios and presentations. The presentations group looked at ways of building translation tools which would map plan/process knowledge from a shared KIF-based representation (see Section 2.7) to preferred system presentations (e.g. Petri Nets [Kiritsis *et al.*, 1998]).

There is still much work to be done on AI planning tools, especially when we consider challenging approaches found in mixed-initiative (see Section 2.3.3) and applied settings

[7] See Appendix F for a sample screenshot of domain editing with the Task Formalism workstation [Tate and Currie, 1984, Tate and Currie, 1985].

[8] An interesting collection of recent papers in this vein were presented at the Fourth International Conference of Artificial Intelligence Planning Systems (AIPS '98) workshop on interactive and collaborative planning.

(see Section 2.3.5). In the following section, we consider the issue that has been called a "highly significant bottleneck in utilising planning systems" [Wang, 1996]: acquiring and maintaining planning domain knowledge.

2.3.7 Planning Knowledge Engineering/Acquisition

> *Back in Section 2.3.6, we introduced the problem of acquiring domain knowledge for a domain-independent planner. While we considered some of the tools and techniques which might be required in Section 2.3.6.1 we delayed an in-depth treatment of this topic until now. In this section, we examine some of the advances in this research area which help to provide acquisition structure and defined methods for this process. Our aim is to present the pragmatic engineering aspects and approaches which would enable discovering, engineering, documenting, and maintaining a set of domain constructs for AI planning.*

The process of acquiring and engineering domain models for use in AI planning involves knowledge-intensive steps. For the most part, these steps are currently considered to be ad hoc and disorganised, at best, for several of the applied planning systems (see Section 2.3.5). In fact, as we have said before, the sources for advice on the process of writing AI planning domain descriptions have been summarised as

> "... it is the most neglected aspect of planning, and there is not an established software engineering methodology to guide this job". [Erol, 1995]

Very recently though, a number of efforts in the AI planning research community have produced a variety of representations, approaches, tools, and architectures for working with AI planning domains[9]. These range from machine learning approaches to the provision of user-based knowledge acquisition tools. This section examines work in this area with a focus on some of the main clusterings. It should be noted that our scope here is mainly limited to generic approaches which would apply to several

[9] See the AIPS'98 workshop papers on Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice [Nunes de Barros *et al.*, 1998].

domains as opposed to knowledge and engineering techniques which were developed for a specific domain area.

We separate some of this work in this area into the following clusterings in order to consider some of the prototypical examples of each: CommonKADS/PSM approaches, formal representations, methodologies, object-centred approaches, domain analysis tools and techniques, learning-based approaches.

2.3.7.1 CommonKADS/PSM approaches

Some researchers believe the best way to chart the various AI planning approaches is with a detailed algorithmic treatment (cf. [Kambhampati *et al.*, 1995]). Nunes de Barros, Valente, and Benjamins presented a differing perspective whereby the focus is on an abstract analysis which highlights the capabilities of the system and the way it represents and uses knowledge [Nunes de Barros *et al.*, 1996].

This knowledge modelling research utilises the CommonKADS [Wielinga *et al.*, 1992, Breuker and van de Velde, 1994] (Knowledge Analysis and Documentation System) methodology which outlines a set of detailed models to be created for a knowledge-based analysis. The KADS/CommonKADS methodology is essentially a generic tool for knowledge acquisition and the building of knowledge-based systems (KBSs). [Breuker and van de Velde, 1994] provides an excellent overview of this approach. In this section we are primarily interested in the application of CommonKADS to planning.

Using CommonKADS, a knowledge engineer uses data about the behaviour of an expert to make design decisions regarding the KBS to be built. This process involves knowledge elicitation, interpretation and formalisation. The components used to represent problem-solving knowledge include: tasks, problem-solving methods (PSMs), assumptions and domain ontologies. One of the most important aspects of the KADS methodology involves reuse of knowledge. This reuse is enabled with the KADS library which contains a number of generic PSMs [Breuker and van de Velde, 1994]. Work on structuring the part of the library that contains PSMs for planning tasks was done by Valente in [Breuker and van de Velde, 1994], pp. 213-230 (see also [Valente, 1995, Nunes de Barros *et al.*, 1996]).

The planning task portion of the library characterises how planners use and structure domain knowledge. This contribution included a definition of a generic task-method decomposition structure along with an identification of the roles knowledge can play in the planning task. The knowledge roles may be static (i.e. they do not change during problem-solving) or dynamic (i.e. contents may change during problem solving). For example, the static roles in planning are presented in Figure 2.4.

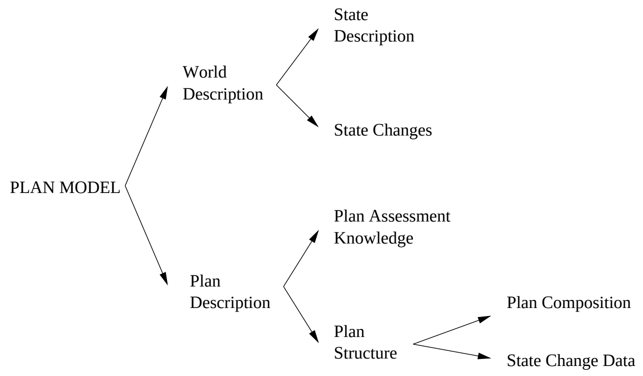

Figure 2.4: Part-of tree of static roles in planning; from [Nunes de Barros et al., 1996, pp. 14]

This CommonKADS planning work facilitates knowledge acquisition (KA) and engineering. For example, a knowledge acquisition tool, TinA (Tool in Acquisition), was developed which uses a library (e.g. CommonKADS plan library) to match assumptions of PSMs with domain knowledge and offer users knowledge-level support for domains [Benjamins *et al.*, 1996].

Other researchers have constructed CommonKADS inference models for planning based on specific systems rather than generic planning approaches. For example, the O-Plan system was modelled using CommonKADS and the models were re-used in the applied task of assignment and management of search and rescue operations by the Royal Air Force [Cottam *et al.*, 1995, Kingston *et al.*, 1996, Cottam and Shadbolt, 1996]. In this work the authors point out that "CommonKADS models are typically developed concurrently with the acquisition of knowledge; initial knowledge acquisition is used to populate higher level models and then these models may be used to document, structure, or guide knowledge acquisition" [Kingston *et al.*, 1996]. This work was also unique in that the authors used the generic O-Plan inference structure as a guide to "critique" the inference structure of the existing domain specific RAF decision process and to spot likely missing steps. In addition to this it has been noted that these knowledge-level descriptions of a planners capability can be used as a means of characterising the domains for which it is suitable [Aylett and Jones, 1996].

Research on the application of KADS to KA for planning domains has also led to new hybrid approaches such as a planning architecture that combines classical and model-based planning technologies [Jarvis, 1997]. In this approach, an object-oriented domain representation is elicited and is used to encode expert application-domain knowledge. Activities and ordering constraints are synthesised by a model-based planner. The output of the model-based reasoning (see Section 2.9) can then be compiled into task refinement schemata and assembled into a complete, interaction free plan by a HTN planner (see Section 2.3.1.3).

Recent research has also brought together a powerful convergence of KADS, KA, and internet-based tools which enable knowledge engineering with distributed "software agents". For example, Crow and Shadbolt have defined the modular Internet-based Multi-agent Problem Solving (IMPS) architecture [Crow and Shadbolt, 1998, Crow and Shadbolt, 1999]. The communication between the agents in this architecture is ontologically underpinned and utilises knowledge level models to integrate information presented in various formats.

2.3.7.2 Formal Representations

To a certain degree, planning research which focuses on formal representations of planning knowledge aides in the process of knowledge engineering and acquisition. For example, Erol's formalisation of HTN Planning [Erol, 1995] has helped to provide a clearer understanding of the various constructs which are available for modelling a domain, such as condition types (cf. [Collins and Pryor, 1993, Tate *et al.*, 1994b]). This aides a knowledge engineer by creating a formal underpinning which may be consulted to clarify precisely the operation of different facets of an HTN planner and how the constructs supported by HTN representational devices affect this operation. This advantage is similar to the understanding provided by clearly defining the inference models and knowledge roles as we described in Section 2.3.7.1. While formal representations provide a more rigorous methodology, they are sometimes considered to be difficult to utilise in practice. Some method of mapping the detailed advice from formal planning work to applied planning techniques has been called for by planning researchers (cf. [McCluskey and Porteous, 1997]).

2.3.7.3 Methodologies

Surprisingly little work has been reported in the literature on methodologies for acquiring and engineering AI planning domains. Unlike software and requirements engineering approaches (see Section 2.11) which have a long history of development models (e.g. waterfall, spiral, etc.) and stages (e.g. specification, design, etc.) an organisation faced with developing an AI planning domain is left without much guidance. There are a few notable exceptions to this generalisation though which we consider here.

Domain capture and modelling has actually been an issue in Edinburgh-based planning research as early as the work on the Nonlin [Tate, 1977] planner. As mentioned earlier, the original O-Plan overall architecture and system design, which dates from 1983, outlined a need for a defined methodology which would guide users performing various roles in the acquisition and analysis of domain requirements for planning [Currie and Tate, 1991]. This planning lifecycle methodology was envisioned as encompassing a set of standardised activities and methods which had well-defined design criteria, techniques, and tools. This was proposed to assist in transforming planning domain development from a craft towards more of an engineering activity. Work looked into adapting the the Controlled Requirements Expression (CORE) methodology [Mullery, 1979, Curwen, 1991] (see Sections 2.3.6.1 and 2.11) for use in planning [Wilson, 1984], but unfortunately this work was set aside. More recently though, a set of guidelines and a checklist for developing O-Plan domain models, referred to as the "TF Method", was added to the TF manual [Tate *et al.*, 1994a]. These components were used in a development outside of the O-Plan team to elicit planning knowledge from the construction industry [Jarvis and Winstanley, 1998, Jarvis, 1997, Jarvis and Winstanley, 1996a, Jarvis and Winstanley, 1996b].

Aylett and Jones described the application of domain independent planning to new domains as a knowledge engineering problem which can be characterised as a system configuration task [Aylett and Jones, 1996]. This work was part of the research on the Advanced Robotic Functional Architecture (ARFA) in which robot planning domains were developed for the Hierarchical Execution Led Planner (HELP). As part of this work, they produced a domain triangle (see Figure 2.5) which can be used to classify different planning domains based on three categories: agents, task, and world. This triangle helps to structure an examination of a particular domain and illustrates the driving attributes which characterise it as well as shows what differentiates it from other domains.

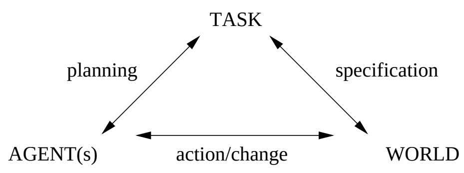

Figure 2.5: Planning domain triangle [Aylett and Jones, 1996, pp. 288]

The EXPECT knowledge acquisition architecture [Swartout and Gil, 1996] can also be considered to provide methodological support via its system-based interactions. EXPECT dynamically forms "expectations" about the knowledge that needs to be acquired by the system and then uses these expectations to interactively guide the user through the knowledge acquisition process.

2.3.7.4 Object-Centred Approaches

In Section 2.3.7.1, we cited work in which an object-oriented domain representation is elicited and is used to encode expert application-domain knowledge [Jarvis, 1997]. This is indicative of a trend in AI planning research which seeks to provide support for constructing planning domain descriptions by adapting methodological steps and notations of the object-oriented community [Jacobson *et al.*, 1992].

Another work [Aylett and Jones, 1996] which we cited earlier (see Section 2.3.7.3) has also made use of object modelling in order to elicit the entities and structure of the target domain. In their example, a simple object hierarchy from the bridges domain was produced in order to understand how the bridge components in the domain interrelated.

McCluskey and Porteus described an approach to engineering and compiling planning domain models which utilises the notion of "lifting" domain representation from the level of the literal to the level of the object [McCluskey and Porteous, 1997]. Once a domain has been described in terms of an object-oriented state transition graph, the author's algorithms compile the diagram into a STRIPS [Fikes and Nilsson, 1971] style action representation. The authors have extended their object-based approach to HTN-style planners as well with their work on OCL-h [McCluskey and Kitchin, 1998].

2.3.7.5 Domain Analysis Tools and Techniques

In Section 2.3.6.2 we looked at various tools for creating and graphically editing plan schemas, Acts, or operators. Some of these tools also support various techniques for performing analytical introspections of domain knowledge. In addition, the work we cited in Section 2.3.6.1 (cf. [Chien, 1996]) has sought to characterise the required checks and tool support for this area. We will also consider some of the other efforts (cf. [Nebel *et al.*, 1997]) which are focused on domain analysis.

The object-centred approach [McCluskey and Porteous, 1997] we described in Section 2.3.7.4 is an excellent example of the emerging domain analysis work. This work defines a set of models and tools which are linked via a coherent method for engineering domain knowledge. These tools include: a syntax, type and consistency checker; goal order, macro and abstraction hierarchy generators; and a random task generator. Sort engineered domain models can then be compiled into operational planning domains.

Another good example of domain analysis tools and techniques is embodied in the research on TIM and STAN [Fox and Long, 1998]. STAN is a planner based on Graphplan [Blum and Furst, 1995] which can take advantage of a number of domain state analysis techniques to improve its performance. These techniques include the automatic generation of fixed-resource invariants and state invariants through the inference of types using the type inference module (TIM). The analysis provided by TIM is planner independent. Their work has also involved the detection of symmetry in a domain which helps to cut down the size of the graph that is constructed [Fox and Long, 1999].

2.3.7.6 Learning-Based Approaches

Another area of research which has had a strong influence on knowledge acquisition and engineering of planning domains is focused on a learning-based approach. Much of this work has been centred around the Prodigy planning architecture [Carbonell *et al.*, 1990, Veloso *et al.*, 1995]. For example, while working with the Prodigy group Gil developed her thesis work (EXPO) [Gil, 1992] on a framework to acquire domain knowledge for planning by failure-driven experimentation with the environment. This describes an approach in which experiments are created and executed in order to validate and adjust domain knowledge. Thus this an example of one way to deal with planning domain knowledge which may be incomplete.

Incomplete domain knowledge has also been tackled in another Prodigy-related research effort into OBSERVER [Wang, 1996]. OBSERVER takes a set of example plans described in terms of the actions in each plan and the state of the world before and after each action. The system examines these examples and generates the preconditions and effects of operator descriptions. This essentially follows a learning-by-doing paradigm and takes a step toward integrating planning, learning and execution.

Other learning-based research has also focused on learning domain control information which can improve the quality of generated plans (e.g. QUALITY [Perez, 1996], Operator Learner [desJardins, 1996], PIPP [Upal and Elio, 1998], ROGUE [Haigh and Veloso, 1998])

2.3.8 AI Planning-Based Process Synthesis

> *At the outset of this chapter we established the point that we are interested in reviewing AI planning and plan representations with an eye towards understanding what research issues are involved in applying this work to integrating the synthesis and management of organisational process knowledge (see Section 1.2). There are a number of projects, rooted in AI planning, which have attempted similar work. In this section we examine some of these approaches and consider problems to be overcome in this technology transfer.*

A good portion of research on AI planning goes toward enabling a single agent (e.g. Shakey [Nilsson, 1984], Xavier [Haigh and Veloso, 1998], etc.) to efficiently plan its actions and to enact them within some specified environment, either real or simulated. AI planning research has branched out to address a number of other scenarios, some of which we described in Section 2.3.5. For example, we discussed the role of AI plan representations within the PLANIT work [Drummond and Tate, 1992] in which plans were generated, communicated and executed across different tools in order to support the organisation. This work was unique in that the representation was the focus as opposed to the actual planning software. In this section though, we are interested in looking at some of the cases where AI planning tools have been used to synthesise new organisational processes either automatically, or semi-automatically (cf. Section 2.3.3).

The idea of using an AI planner to help synthesise and structure the activities between a generic collection of performing agents (e.g. an organisation, a department, a business unit, etc.) actually goes back quite a way in the history of this field. For example, Fikes described a commitment-based framework [Fikes, 1982] for this purpose. This approach advocated the formation of commitments from one agent to another which could be used later to highlight the dependencies between agents.

Some of the approaches reported in the literature have focused on tackling synthesis within specific process domains such as: software development processes, manufacturing steps, chemical plant procedures and military or defence-related operations.

- Software development processes: Huff and Lesser developed a constraint-based language called GRAPPLE which was used to model software development processes as a set of goals, subgoals, preconditions, constraints, and effects. With GRAPPLE, they would construct process models from two fundamental components: a set of process steps and a set of constraints on how those steps can be selected, ordered and applied. The value in this approach was in the rich representation of the internal structure and dependencies. Planning techniques were also used in Agora [Bisiani *et al.*, 1988] which provided a domain-specific planner for tasks relating to heterogeneous, parallel systems.

- Manufacturing steps: Process plans are machining instructions which are used to manufacture mechanical parts. A range of constraints are involved in specifying the detail of the planned steps. Researchers have been working on applying AI planning to this task as well. For example, the work on the IMACS (Interactive Manufacturability Analysis and Critiquing System) project

[Nau *et al.*, 1995, Gupta *et al.*, 1998] has defined a method whereby products are broken down into a set of features which are then mapped to a sequence of operations which can create it. Another feature-based approach is outlined in the Arizona State University Feature Testbed (ASUFTB) [Batchu *et al.*, 1995] manufacturing system. This system supports an iterative and interactive approach which helps the user to focus on which parameter to improve along with where and how to modify the plan.

- Chemical plant procedures: AI planning has been applied to the design of operating procedures for chemical plants. For example, the chemical engineering planner (CEP) was developed as part of the EPSRC funded INT-OP program [Aylett *et al.*, 1997, Aylett *et al.*, 1998]. Part of this work also detailed an architecture which integrated the application of AI planning and techniques from the operating procedure synthesis literature [Soutter, 1997]. This work showed how to address issues in valve sequencing and safety. Additional research has looked into planning for a monitoring and control system extended by knowledge-based features in order to realize automation tasks and to relieve system operators in the chemical industry [Jantke *et al.*, 1996]. Some of the main problems addressed concerned process safety and protecting from dangerous situations.

- Military/Defence-related operations: Probably one of the most researched domains for the application of AI planning to process synthesis involves military and defence-related processes. Much of this work has been part of the ARPA-Rome Laboratory Knowledge-Based Planning and Scheduling Initiative (ARPI) [Fowler *et al.*, 1996] which we introduced in Section 2.3.3. For example, the System for Operations Crisis Action Planning (SOCAP) [Bienkowski, 1996] was developed and used in an integrated feasibility demonstration which had a focus on operations and transportation planning for small-scale defensive military. SOCAP integrated advanced generative planning, temporal and case-based reasoning, and scheduling techniques to generate these military operations plans. Another example of a defence-related application is the Automated Scheduling and Planning Environment (ASPEN) [Fukunaga *et al.*, 1997] which was developed for the spacecraft mission planning process at the Jet Propulsion Laboratory (JPL).

> The main elements of ASPEN plans include activities, resources, states, temporal constraints and reservations.

All of these examples serve to illustrate the maturing use of AI planning software in synthesising processes for a range of domains. One of the common themes running through them is "integration". The capabilities of an AI planner are only useful when they can be integrated with other reasoning techniques (e.g. valve sequencing in chemical plant designs or process plan evaluation in IMACS). In addition to this, we can see common issues related to the lack of knowledge engineering and acquisition guidance (see Section 2.3.7) for building domain knowledge. This viewpoint was expressed in Curtis' review of process modelling techniques

> "The ability of a constraint-based planning system ... for developing effective process plans depends on the success of its designer in coding the knowledge about the environment and the goal hierarchies of the process into the components of [the target language]." [Curtis *et al.*, 1992]

2.3.9 Moving Forward

> *In this final section covering domain-independent AI planning we examine some of the challenges which the field is faced with as it moves forward. Additionally, we consider a driving perspective which envisions the application of AI planning in an integration role. Our goal is to show that this integration perspective appears to partially address many of the cited challenges.*

The premier conference for planning research is the International Conference on Artificial Intelligence Planning Systems (AIPS) which is held every two years. At the most recent gathering (June 1998, Carnegie-Mellon University) the conference chair, James Allen, laid out four challenge areas for the field in the opening address

- More work on expressive representations
- Relationship between planning and execution

- Attacking real applications
- Human-Computer Interaction

Work continues on incorporating more expressive representations (e.g. conditional effects in Graphplan [Anderson *et al.*, 1998]) and on the relationship between planning and the uses of a plan, such as its execution (e.g. O-Plan work on planning and execution [Reece and Tate, 1994, Reece, 1994, Drabble *et al.*, 1997a]). As we showed in Sections 2.3.5 and 2.3.8, several projects have been working on applied uses and in Section 2.3.3 we considered some of the work currently tackling human-computer interaction.

These four areas serve as reminders that, in order to succeed, AI systems which are deployed in the real world require integration into the environment in which they operate. Plans from AI planning systems need to be able to be interleaved with information which may exist in other tools or databases. A planner's input and output must be in a form that is both expressive and easily understood by users. These views were unified in McDermott and Hendler's perspective for a possible future of AI planning

> "... view general-purpose planning as providing an architectural framework for combining results from more specialised systems. That is, the general-purpose system provides a common ground for talking about plans, transformations on plans, and thereby provides a protocol for specialised reasoning algorithms to plug into." [McDermott and Hendler, 1995]

It is possible that such an architectural framework might go a long way toward addressing the challenge areas given above. This framework must be flexible enough to handle expressive representations, be capable of supporting interoperability with execution and mixed-initiative tools and must work for real, applied scenarios.

2.4 Representations in Logic

The purpose of this section is to succinctly present some aspects of the use of logic in representing knowledge about plans and actions. We cite some of the major approaches in this area and consider the relationship that this *formal system of representation has to modern, constraint-based views of plan models.*

The use of logics in AI [Genesereth and Nilsson, 1988] and, in particular, in AI planning has a long history as we mentioned in Section 2.3.1.1 with the work on QA3 [Green, 1969] and the situation calculus [McCarthy and Hayes, 1969]. The situation calculus is a *first order language* which was designed for reasoning about actions. First order languages are based on first order predicate logic (FOPL) [Chang and Lee, 1973, Loveland, 1978, Gallier, 1986] which has a well-defined semantics and is arguably [Davis, 1990] the most important and commonly used logical system.

FOPL representations expressed in situation calculus identify *situations* which are snapshots of a world, *fluents* which are time-varying properties (i.e. the values of these properties may be different in separate situations), and *actions* which transform one situation to another by possibly changing the value of fluents. A fluent, f, is said to "hold" in some situation, s, which is expressed with an atomic formula, holds(f,s). A function term, result(a,s), is used to obtain the situation which is produced when this action is performed in situation s. Effect axioms are used to represent the effects and preconditions of actions. Over time, the situation calculus has been extended in a number of ways to deal with concepts such as: concurrency, non-instantaneous actions, conditional actions [Gelfond *et al.*, 1991]; concurrent actions and the notion of independence among actions [Lin and Shoham, 1991]; and complex actions (i.e. non-primitive, primitive actions) [Gruninger and Pinto, 1995].

2.4.1 Logic Programming

The original situation calculus had little support for the representation of time. Pinto and Reiter proposed an axiomatisation of an extended version of the situation calculus for temporal reasoning in a *logic programming* framework [Pinto and Reiter, 1993a, Pinto and Reiter, 1993b]. Logic programming is a programming language paradigm in which logical assertions are viewed as programs. Many such logic programming systems have been developed, but the most popular one is Prolog [Clocksin and Mellish, 1981].

A Prolog program is written as a series of logical horn clause assertions which are reasoned over using resolution theorem proving.

Another important logic programming approach to reasoning about temporal aspects of actions and plans is the event calculus (EC) [Kowalski and Sergot, 1986]. The EC can be used to represent the occurrence of events, the properties that events initiate and terminate, and the maximal time periods over which these properties hold. Two functions are used to deal with time periods: before(a,f) and after(a,f) where a is an action and f is a fluent. The term after(a,f) names a time period. The sentence, Holds(p) expresses that a relationship which is associated with p (e.g. after(pickup(Block),holding(Block))) holds for the time period p. Various variants to EC have been introduced (cf. [Sadri and Kowalski, 1995]) some of which employ time points rather than time periods. As with situation calculus, some of this work has been utilised in planning systems (abductive planners [Eshghi, 1988, Missiaen *et al.*, 1995, Shanahan, 1997b]). Some systems, such as REACTIVE PASCAL can utilise *either* situation calculus (as was done in GOLOG [Levesque *et al.*, 1997]) or event calculus as "background theories" for temporal reasoning [Quintero, 1996].

2.4.2 Advanced Logics and Constraints

In Section 2.3.1.1 we mentioned the frame problem [Hayes, 1973, Shanahan, 1997a] which appears when reasoning about action. A specific group of logics have been developed to address this problem: nonmonotonic languages (see [Davis, 1990]). These efforts include research into default logic [Reiter, 1980] and circumscription [McCarthy, 1980]. Other logics, e.g. modal logics [Hintikka, 1962], have been developed to reason about beliefs or "modes" in which a statement may be true. Modal logics allow us to talk about the truth of a set of statements not only in the current state of the real world but also about their truth or falsehood in the past or future (i.e. temporal logics) and about their truth or falsehood under circumstances that might have been, but were not (i.e. conditional logics) [Rich and Knight, 1991].

Temporal logics are of particular interest to planning researchers (cf. [McDermott, 1982, Allen, 1984a]). Over time, many approaches have developed. Hayes compiled a catalogue of temporal theories which noted many of the possible ontological choices available for the representation of time [Hayes, 1996] and Orgun and Ma have provided a logic programming overview of the application of temporal and modal logics [Orgun and Ma, 1994]. For example, one of these works reviewed included Chronolog, a temporal version of Prolog [Orgun and Wadge, 1992].

Logic programming in systems such as Chronolog can be considered to be addressing a temporal constraint satisfaction problem (TCSP) [Schwalb and Vila, 1996]. Schwalb and Vila have elaborated this notion in their survey of temporal constraints [Schwalb and Vila, 1998]. Recalling back to Section 2.3.1.4 we can see that this notion of treating time as a class of "temporal constraints" is one which has been adopted by the AI planning community. In fact, as part of Dave Joslin's ARPI work[10], he proposed that a sorted first order logic [Cohn, 1985, Walther, 1985, Davis, 1990] could be used to represent a range of planning constraints, including temporal constraints. This language could act as an interface between a planner and a scheduler and a compiler could translate these constraints into a CSP to be efficiently solved.

2.5 Design Rationale

In this section we return to the issue of planning rationale which we presented in Section 2.3.2. In particular we are interested in the representation and communication of planning decisions. As we shall see there has been work which relates planning to design. The design community has a subfield which has researched the expression of elements related to design decisions. We will briefly point out some of the work in this area and note its relevance to AI plan representations.

Recent work contributing toward international standardisation for process and plan interchange have produced new perspectives on plan representations. One of these perspectives relates *plans* to *designs* [Tate, 1996d]. Tate defines a plan as a specialised type of design where a "design for some artifact is a set of constraints on the relation-

[10] This is from personal communication in May 1996 while Dave Joslin was working at the Computational Intelligence Research Laboratory (CIRL) (University of Oregon).

ships between the entities involved in the artifact". A *plan* constricts this definition by specifying that the entities are agents, their purposes, and their behaviour.

Planning can then be considered to be a specialised type of design activity. Designs or plans are created by an agent or group of agents placing constraints on the developing artifact. The application of a constraint typically arises from a design decision that was made (e.g. the walls must be 4 in. thick, use expansion A rather than expansion B, etc.). We can think of these activities as repeatedly making design decisions that continually transform the artifact until it embodies the requirements necessary to enact the solution. In real-world scenarios for both planning (see Section 2.3.5) and design we often have a need to understand the reasons behind these decisions (see decision rationale on page 36).

Designers cooperate by sharing rationale and often need to look behind the artifact to understand the deeper meanings behind the constructs. The research that has addressed this need in the design community is called design rationale (DR) (cf. [Moran and Carroll, 1996]). A design rationale is a representation of the reasoning behind the design of a system. It is essentially the explicit recording of the issues, alternatives and justifications that were relevant to elements in the design of an artifact. Examples of design rationale implementations include: QOC [MacLean *et al.*, 1991], DRL [Lee, 1990], gIBIS [Conklin and Begeman, 1988]. Each DR implementation offers some trade-off between [Lee and Lai, 1991]:

- expressiveness
- human usability
- computer usability

This trade-off can be expressed in the way that these notations or languages vary on a set of cognitive dimensions (e.g. premature commitment, viscosity, hidden dependencies, role expressiveness) [Buckingham Shum, 1991a, Buckingham Shum, 1991b]. In reviewing these issues it is important to remember that ultimately the goal is to support design activities during the lifecycle of the design. This support addresses the design process in a number of ways. For example, a representation that includes design rationale has been shown to lead to a better understanding of the issues involved [Conklin and Yakemovic, 1991]. MacLean et al. list two major benefits from design rationale representation [MacLean *et al.*, 1991]: an aid to reasoning and an aid to communication. A simple outline of the QOC notation which they used is given in Figure 2.6.

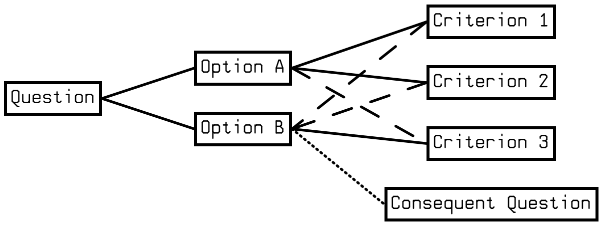

Figure 2.6: QOC, semi-formal notation to represent a design space. Dashed arcs between options and criteria denote negative influence whereas solid arcs indicate positive influence (i.e. arguing for or against an option).

In more recent work, a series of empirical studies have shown that this QOC approach provides most support when elaborating poorly understood design spaces, but can be a distraction when evaluating well constrained design spaces [Buckingham Shum *et al.*, 1997]. All of these benefits: understanding, reasoning, and communication apply to several stages in the lifecycle of a design or plan. While the focus is usually on DR's contribution to the initial construction of the design, there is also rich support for the maintenance and reuse of the design as well. An artifact lacking rationale can often be hard to understand when revisited at a later date or by another agent who wasn't involved in the original design process. Changing requirements or environments may require incremental modifications to the design or to plans.

2.5.1 Putting DR to Use

A number of projects have benefited from the incorporation of design rationale into their approach. For example, Ballinger et al. reported on changing design factors that necessitate the consideration or reconsideration of various issues in the design of a chemical plant [Bañares–Alcántara, 1991, Ballinger *et al.*, 1993]. They utilised an IBIS-type [Conklin and Begeman, 1988] structure to connect the new alternatives, or positions to the issue. The agents then participated in the generation of criteria that would lead to a series of choices. Some of the strengths and weaknesses of this IBIS DR approach were considered in Chung and Goodwin's work on an integrated design information system (IDIS) [Chung and Goodwin, 1994]. The contributions of their work also included the identification of a need to monitor the temporal integrity of a design argument along with establishing a method to *automatically record design changes* from within a design tool, viz. AutoCAD. This notion of automatically acquiring DR during the design process is also outlined in the DARPA RaDEO work on SRI's rationale acquisition framework (RAF) which provides tools and methods that enables human designers to extend and modify rationales.

Some of the approaches towards putting DR to use have begun to consider the possible relationship that design rationale has to planning. For example, research into an agent-based project management system (ProcessLink) utilised and extended a general model of design change propagation (Redux) which makes design rationale active by tracking several aspects of a plan's validity and informing agents when it changes [Petrie *et al.*, 1999]. This work is indicative of a move toward distributed integrated project management (DIPM). The authors describe DIPM as "an extreme form of process coordination in which design, planning, scheduling and execution are interleaved across distributed organisations and engineering disciplines as well as computer tools". This notion of integration is a recurrent theme which we have encountered in a number of review areas including process synthesis (see Section 2.3.8) and in the challenges facing AI planning research (see Section 2.3.9). In the following section we consider some of the more general work which has attempted to address integration of information systems.

2.6 Integration of Information Systems

The management of data relating to organisational processes is also a concern to researchers who are attempting to integrate the information which is developed, modified, and required across a range of information systems. We cite some of these enterprise-wide efforts with an eye towards tools and techniques which show some promise in this endeavour.

Most enterprises rely on information technology (IT). Tools and applications are developed to automate and assist in various tasks such as modelling and design, simulation and scheduling, data storage and retrieval. One of the most important industry-wide movements in IT is the integration of these heterogeneous systems which may be distributed throughout the organisation (cf. [Mertins and Schmidt, 1998]).

One example of a project involved in this integration effort is the KRAFT (Knowledge Reuse and Fusion / Transformation) project [Gray *et al.*, 1997]. KRAFT's primary goal is to define and build an architecture in which various kinds of middleware agents cooperate to locate, combine and refine knowledge and data to solve a given problem. In this work, ontologies (see Section 2.7.1) play roles in helping locate and translate relevant knowledge as well as being utilised as background knowledge.

Ontologies and standards are being deployed in a range of enterprise integration efforts. For example, information modelled in a case tool such as AIAI's HARDY meta-case tool[11] could be exchanged with other design tools via the Case Data and Interchange Format (CDIF) [Ernst, 1997]. CDIF is a standard for the exchange of case data information which also outlines integrated meta-model areas, such as the project management planning and scheduling subject area [Navarro, 1996].

In Section 2.3.4.2 we introduced some other interchange formats which are aimed at integrating systems in various environments. For example, the PIF work is interested in supporting a wide variety of process tools such as process modellers, workflow software, planners, process simulation systems, etc. [Lee *et al.*, 1998b] and the PSL work [Schlenoff *et al.*, 1996] is tackling similar issues in a manufacturing setting. The ontologies being built for both of these efforts are partially based on other enterprise integration efforts such as the Toronto Virtual Enterprise (TOVE) [Fox *et al.*, 1993] and the Edinburgh Enterprise work [Fraser and Tate, 1995, Uschold *et al.*, 1998].

Researchers are also looking into internet and web-based integration methods. For example, the RDF/XML [Lassila, 1998] and SHOE [Luke *et al.*, 1997] efforts are looking into ways of enriching HTML pages with ontologically underpinned terms which would assist in automated processing of HTML page data. Other efforts have looked to providing CORBA IDL specifications [Madni and Mi, 1997] which would provide distributed, well-defined interfaces that could support integrated process modelling and

[11] See http://www.aiai.ed.ac.uk/project/ for information on the HARDY project at AIAI.

intelligent workflow in enterprises.

In summary, it appears that two of the most common methods for establishing the integration of information systems involves the development of shared ontologies and some accepted standards which would help to define an acceptable interchange format. This overlaps with the research into knowledge sharing which we will examine in Section 2.7.

2.7 Knowledge Sharing

In Section 2.3.4 we looked at efforts aimed at a specific class of knowledge sharing which involves the exchange of plan and process information. Many of these projects have benefited from a variety of general techniques and approaches which enable sharing and reuse of knowledge. In this section we look at some of this technology with a focus on ontologies and grammatical models.

Some of the most influential knowledge sharing projects reported in the literature involved work executed under the umbrella of the DARPA/AFOSR/NSF funded Knowledge-Sharing Effort (KSE) [Neches *et al.*, 1991]. The KSE initiative was focused on the development of a technical infrastructure to support the sharing of knowledge among systems. This work was partially motivated by a set of identified impediments to sharing and reuse which included

- *Heterogeneous representations*: There is no single knowledge representation that is best for all problems, nor is there likely to be one.
- *Dialects within language families*: Even within a single family of knowledge representation formalisms it is often the case that knowledge has been encoded in different dialects.
- *Lack of communication conventions*: We lack the conventions and standards required for systems to intercommunicate knowledge.
- *Model mismatches at the knowledge level*: Different primitive terms are used and systems lack a shared vocabulary and domain terminology.

Various KSE working groups were formed to tackle some aspects of each of these stumbling blocks. For example, the KQML protocol [Finin, 1992] we cited in Section 2.3.4.1 was developed to provide a solution to the *lack of communication conventions*. The problem of *heterogeneous representations* was addressed from a translation approach with the provision of a shared interlingua. The language proposed to express the shared knowledge was KIF, the Knowledge Interchange Format [Genesereth, 1991, Genesereth *et al.*, 1992]. A central operational requirement for KIF was that it enable practical means of translating declarative knowledge bases to and from typical knowledge representation languages.

Some representations, such as description logics (cf. [Calvanese *et al.*, 1998]) were produced by the Knowledge Representation Systems Specifications (KRSS) working group to enable the definitional expression of concepts (similar to KL-ONE systems such as CLASSIC and LOOM) to address *model mismatches at the knowledge level*. This set of defined terms and concepts is often referred to as an ontology. Gruber provided an alternative format called Ontolingua [Gruber, 1993] for representing ontologies whose syntax and semantics were based on KIF.

2.7.1 Ontology

The concept of "ontology" is drawn from philosophy in which it is used to indicate a systematic theory about existence. The use of this term in computational settings (e.g. in information systems, or artificial intelligence applications) tends to vary depending on particular needs and perspectives. On one extreme, people may refer to an ontology as simply a lexicon of terms for a particular application (e.g. for an automotive domain we might have: WHEEL, BODY, ENGINE, BRAKE, etc.) while on the other end of the spectrum they may mean a particularly rigorous set of logical axioms which provide detailed terms and definitions. See [Uschold and Gruninger, 1996] for an overview of this range of formality and for an introduction to this field. Gruber defines an ontology as

> "An ontology is a vocabulary of terms (names of relations, functions, individuals) defined in a form that is both human and machine readable. An ontology, together with a kernel syntax and semantics, provides the language

> by which knowledge-based systems can interoperate at the knowledge-level." [Gruber, 1993]

As this definition implies, ontologies typically need to be referred to and inspected by people. People review parts or all of the ontology in order to align themselves with the "shared understanding" of the set of concepts (e.g. during translation writing, or in clarifying assumptions). Ontological work has been conducted on both large scales (cf. CYC [Guha and Lenat, 1990, Guha and Lenat, 1994, Lenat, 1995] which uses it's own custom language called CYC-L) and for smaller scales such as a domain specific ontology for disturbance diagnosis and service recovery planning in electrical networks [Bernaras *et al.*, 1996].

2.7.2 Pluggable Grammars

Several projects involved in knowledge sharing and communication have developed highly *flexible* methods for exchanging knowledge. For example, in his thesis work on a capability description language (CDL) [Wickler, 1999], Wickler described a method for dealing with the *classic trade-off* that can be found in knowledge representation and reasoning: expressiveness versus efficiency. He defined *flexibility* as

> "A knowledge representation language is **flexible** if it allows the knowledge engineer to choose a compromise regarding a certain trade-off at the time of knowledge representation rather than having to adopt a fixed compromise prescribed by and designed into the representation." [Wickler, 1999]

In his approach, the expression of capability constraints are linked to a language module which provides the grammatical definition of the constraint content. A similar grammar-based approach has also been used in the U.S. military planning research community. For example, there has been work to use verb/noun phrase grammars to represent various expressions of plan objectives and activities [Hess, 1996, Kingston *et al.*, 1997, Drabble *et al.*, 1997b]. In addition to these, part of the INSPECT work on air campaign plans focused on the development of a BNF (Backus-Naur Form) for flexibly expressing objectives [Valente *et al.*, 1996]. There are also examples of using this grammar-based method [Pentland, 1994] to define and configure various aspects of organisational processes. This work was part of the efforts on redesigning organisational processes (see Section 2.12) based on the Process Handbook [Malone *et al.*, 1993].

2.8 KBS/Ontology Building Methodologies

In Section 2.3.6, we cited work which highlighted the problem of "how to get domain knowledge into a planner?". We briefly discussed relevant work on tool support in Section 2.3.6.1 and also looked at various approaches to this research issue in Section 2.3.7. Some of this work is linked to a more generic corpus of research on ontological and knowledge-based engineering methodologies. We will look at examples of work in this area with an emphasis on approaches which may aid in engineering process knowledge for an AI planning system.

Several texts have been written on the subject of engineering knowledge for knowledge-based/expert systems (cf. [Hayes-Roth *et al.*, 1983, Prerau, 1990], [Liebowitz and De Salvo, 1989, Giarratano and Riley, 1994], [Guida and Tasso, 1994, Breuker and van de Velde, 1994]). Many of these texts address the methods and lifecycle of development along with a presentation of how the technology works (knowledge representation and inferencing). Several significant systems have been produced which illustrate both broad tool-based support and a comprehensive methodology (cf. VITAL [O'Hara *et al.*, 1992, Domingue *et al.*, 1993] and its predecessor KEATS [Eisenstadt *et al.*, 1990, Motta *et al.*, 1991]). While it is possible to use many of the standard approaches discussed in these works, some of the early generic texts discussed requirements which are more directly related to engineering knowledge for basic AI planning approaches [Hayes-Roth *et al.*, 1983].

In one example, [Stefik *et al.*, 1983] noted that HTN-style planners deal with "no fixed sequence of sub-problems" by utilising an abstract search space (see Section 2.3.1.3) and many partial-ordered planners address "interacting subproblems" via constraint propagation and least commitment (see Section 2.3.1.4). This suggests that the engineering and acquisition of knowledge is often linked to or specialised for the *capabilities* of the system. The construction of models for a problem domain has been recognised by the KBS community as an important component in the overall task of knowledge acquisition for expert systems relative to some specific problem-solving framework [Davis and Bonnell, 1991]. For example, the issues surrounding knowledge acquisition (KA) for the ONOCIN planner for various domains yielded the development of specialised knowledge acquisition tools such as Opal and Protégé [Musen, 1989, Eriksson *et al.*, 1995].

Protégé included a model of problem solving using a method of episodic skeletal-plan refinement in which knowledge engineers assemble a model using a library of smaller building blocks, called problem-solving mechanisms (PSMs). This connects back to the KADS-based research we discussed in Section 2.3.7.1 which also uses PSM models of planning and plan tasks. Protégé is also a meta-level program which generates knowledge-acquisition tools tailored for classes of application tasks. This architecture allows knowledge engineers to represent static, reusable domain knowledge as explicit *ontologies* of concepts and relationships.

2.8.1 Ontological Support

In Section 2.7.1 we discussed the role of ontologies within the context of knowledge sharing. Ontologies are also being put to use in engineering knowledge as we pointed out above and in the work on KRAFT (page 71). Another excellent example of this is the Structured Process Elicitation and Demonstration Environment (SPEDE) which is a "methodologically grounded toolset that provides support for Business Process reengineering" [Cottam *et al.*, 1998] (See Section 2.12). In this work the authors show how ontologies can be used to guide the KA process. Ontological support is also present in tasks such as verification and reuse, e.g. checking artifacts which are required to adhere to ontological definitions [Kalfoglou and Robertson, 1999]; and providing reusable knowledge for businesses [Jin *et al.*, 1998].

With the growing interest in developing ontologies for use in various domains, researchers have also produced generic methods for constructing domain ontologies. For example, the work on Methontology [Gómez-Pérez *et al.*, 1996, Fernández *et al.*, 1997] outlines an approach toward specifying ontologies at the knowledge level using a series of intermediate representations.

In addition to the examples of PSM and ontologically-based approaches we have presented here, other possible sources of generic techniques toward knowledge modelling and domain knowledge engineering for AI planning include work on *generic tasks* [Chandrasekaran, 1987], *object-oriented methodologies* [Jacobson *et al.*, 1992, Booch *et al.*, 1998], *role-limiting methods* [Marcus (ed.), 1988] and *components of expertise* [Steels, 1990].

2.9 Model-Based Reasoning

In Section 2.3.7.1 we looked at some of the approaches to knowledge acquisition of plan domain models. In that review, we cited work which has utilised model-based reasoning alongside classical AI planning techniques. In this section we will look at some additional benefits of model-based reasoning in synthesising and managing organisational processes.

Model-based reasoning approaches (see [Hamscher *et al.*, 1992]) grew out of work on diagnostic tasks. Representations of the structure and behaviour of domain components have been used to detect faults and to infer minimal sets of components which explain problems associated with observed measurements. For example, de Kleer and Williams described the Generic Diagnostic Engine (GDE) [de Kleer and Williams, 1986] which could be used to find multiple faults in interconnected logical and arithmetic circuits.

One of the problems which faces potential end-users of domain-independent planning is the difficulty in constructing adequate and provably consistent domain models. As pointed out in the charter of the PLANET Network of Excellence In AI Planning, Knowledge Acquisition Technical Coordination Unit[12], this is particularly important when planning in large and safety critical application domains. One approach to this problem, which we cited in Section 2.3.7.1, combines classical planning and model-based reasoning technologies [Jarvis, 1997]. In this case, the output of model-based reasoning is compiled into an HTN-style domain in order to synthesise new plans.

[12] The home page of the PLANET European Network of Excellence in AI planning can be found at: http://planet.dfki.de/.

Model-based reasoning has also been directly applied to the synthesis and evaluation of organisational processes as well. For example, the Comet system [Nado *et al.*, 1996] has been applied to a financial auditing domain. Comet can be used to create a "hierarchical flowchart model" that describes the intended processing of business transactions by an accounting system and the operation of its "internal controls". Model-based reasoning is used to automatically analyse the effectiveness of the controls in detecting potential errors[13]. Part of this approach overlaps with work on modelling business processes and business process reengineering which we address in Section 2.12.

2.10 Agent Architectures and Environments

In discussing a mixed-initiative approach to planning (see Section 2.3.3), we introduced research which helps to support multiple agents participating in the development and management of organisational processes/plans. This issue was also partially addressed by the work we reviewed on knowledge sharing and shared representations (see Sections 2.3.4.2 and 2.7). We will briefly present some research into agent architectures and environments which underpins or complement these efforts with contributions to areas such as control, message protocols and authority.

We introduced KQML [Finin, 1992] in Section 2.3.4.1 which is the *agent protocol* advocated by the Knowledge-Sharing Effort (KSE) [Neches *et al.*, 1991]. KQML defines various "performatives" which label messages that are sent between agents and to "brokers" which aid in locating other agents and managing agent communication. The content of KQML messages vary, depending on the particular performative being used. Research into various types of middle agents which help to coordinate efforts have been introduced in the literature (cf. [Decker *et al.*, 1997]). These middle agent roles range from blackboard-style support (e.g. [Hildum *et al.*, 1998]) to more complex forms of arbitration.

We also looked at the ProcessLink agent-based project management system [Petrie *et al.*, 1999] in Section 2.5.1. This distributed, integrated project manage-

[13] Comet is used by Price Waterhouse auditors world-wide. A similar set of systems, WinProcess/WinSmart, have been implemented at Arthur Andersen, LLP.

ment approach views collaboration from a design-based perspective. Message-passing between agents is linked to dependencies between the plan and various design elements. While there is a simplified notion of *authority* in which decision makers are solely responsible for their own decisions along with a superordinate "design manager", other researchers have looked into more complex agent authority issues (cf. [Tate, 1993a]).

The notion of *control* is a central concern to many agent-based approaches. For example, the Distributed Intelligent Control and Management (DICAM) project, which is closely related to the NASA/NBS reference model (NASREM) [Albus *et al.*, 1989], advocates a controller reference architecture which includes a "collection of semi-autonomous interconnected controllers" [Hayes-Roth *et al.*, 1992]. These include both domain and meta-level control systems[14].

In addition to this, researchers have also been looking into novel platform support for computer supported cooperative work (CSCW) [Baecker, 1993, Rashid and Helal, 1997]. For example, Tennison's thesis work [Tennison, 1999] provides an example of support for the process of distributed collaborative work based on methodologies of distributed knowledge engineering. This approach utilises a Collaborative Virtual Environment (CVE), viz. a Multi-user {Domain|Dungeon} Object-Oriented (MOO) environment.

Multi-agent systems are present in many AI planning approaches as we indicated with the work on mixed-initiative approaches. Some of the well-known examples also include Konolige's earlier work [Konolige and Nilsson, 1980, Konolige, 1982] and more recent research into O-Plan [Tate *et al.*, 1994c, Tate *et al.*, 1998a], the MPA agent architecture [Wilkins and Myers, 1998] and the University of Washington's intelligent softbots [Etzioni *et al.*, 1993, Weld, 1996, Etzioni, 1997].

As we can see, building multi-agent architectures and environments are complex endeavours. They are prone to a number of pitfalls [Wooldridge and Jennings, 1998] beyond those of traditionally engineered, distributed software approaches. Work is underway to establish methodologies for developing these types of systems (cf. [Wooldridge *et al.*, 1999]) as well as adapting techniques to support reasoning and ne-

[14] These are coming to be known as "fractal architectures" since the individual components can be combined into organisations in various ways. For example, the organisational structure can be hierarchical, peer to peer, recursive, etc.

gotiation [Parsons *et al.*, 1998] amongst agents.

2.11 Requirements Engineering

In section 2.3.2 we pointed to the similarities between the process of engineering requirements for software systems and eliciting and managing constraints for organisational designs/plans. We also cited work which outlined the possible adaptation of a requirements methodology, CORE, for use in structuring the methods of engineering planning domain requirements/constraints in Sections 2.3.6.1 and 2.3.7.3. In this section, we will present some approaches to requirements engineering, with an emphasis on overviewing the work products of the CORE methodology.

Requirements engineering is "the process of discovering, documenting and managing the requirements for a computer-based system" [Sommerville and Sawyer, 1997]. The goal of the process is to reasonably approximate a definition of the customer's needs and expectations for the behaviour of the system. A number of approaches, techniques, and systems have been created to address various aspects of this task[15]. This includes work on viewpoint management and stake-holder analysis [Easterbrook and Nuseibeh, 1996, Kotonya and Somerville, 1996, Finkelstein *et al.*, 1994], as well as work on various methodologies, techniques, and guidelines [Sommerville and Sawyer, 1997, van Lamsweerde and Letier, 1998] for eliciting, recording, and managing requirements.

Some researchers have approached the expression of these requirements using AI-based knowledge representations. For example, Greenspan et al. described the Requirements Modelling Language (RML) [Greenspan *et al.*, 1982, Greenspan, 1984] which can be used to capture rich content including the specification of entities, activities, and assertions. RML was melded with a structured analysis language, SADT which is used to build a structured lexicon of relevant terms [Borgida *et al.*, 1985]. Other researchers argue that requirements engineering needs to be viewed as social engineering. The

[15] See [Davis, 1990] for an introduction and an excellent compilation of some of the earlier work. See the Requirements Engineering Network, RENOIR, for more recent research, http://www.cs.ucl.ac.uk/research/renoir/.

modelling language developed in the ORDIT project [Dobson *et al.*, 1994] is, in contrast, visual in nature in order to diagrammatically represent and reason about the impact that a software system may have on an organisation.

In discussing Integrated Project Support Environments (IPSEs), Snowdon and Warboys drew an analogy between "a company producing an information system" and an "automated manufacturing plant" [Snowdon and Warboys, 1994]. In both cases, a collection of humans and/or machines are meant to enact some process model subject to certain requirements. These requirements may be functional/non-functional for the information system and the manufacturing plant may be required to follow material or safety guidelines and constraints. In both cases, similar techniques for eliciting the domain requirements may be employed.

As we stated in Section 2.3.7.3, research was conducted into a requirements engineering methodology which could be adapted for use in eliciting AI planning domain constraints. The Controlled Requirements Expression (CORE) was proposed for structuring these domain management activities. It was hoped that an adaptation of this method, combined with experience in working with the Task Formalism, could help to drive the development of planning domains in a more reliable fashion.

2.11.1 CORE: A Software Requirements Methodology

COntrolled Requirements Expression (CORE) was a method developed by British Aerospace (Warton) and Systems Designers, Ltd. in the late 70's [Mullery, 1979]. Over time, the method has evolved and CORE now provides techniques for requirements capture, analysis and specification [Curwen, 1991]. The method can be used to partition problems into manageable modules which can be assessed using CORE analytical techniques. This helps to ensure that the requirements for a specification are complete and consistent. Some of the strengths of this methodology include decomposability of requirements and traceability mechanisms between different levels of requirements.

The CORE specifications are expressed in terms of graphics (as in ORDIT), structured text and specialised notations. These resultant requirements models start from operational requirements which influence functional requirements and, in turn, impact implementation requirements (with non-functional requirements acting as functional and implementation constraints). Viewpoints are used as logical partitionings of the system under consideration.

Connecting domain aspects to their underlying requirements may assist in managing domain modifications which are the result of changing needs of an organisation. Clearly defined roles and responsibilities at the requirement level help to organise the activities at the domain level. This could address one of the major impediments which has prevented the adoption of AI planning tools and techniques in applied settings: a lack of organisational context.

2.12 BPR/Knowledge Management

We conclude our survey of related research areas with an overview of approaches to reengineering business processes and managing enterprise knowledge. In this section we are interested in providing examples of these approaches and in understanding what improvements might be made by incorporating experience and techniques from AI planning.

In their seminal work on organisational processes, Hammer and Champy suggested that the traditional methods of task-based work organisation were obsolete [Hammer and Champy, 1994]. Three fundamental factors were demanding new ways of thinking in terms of organisational processes: *customers*, *competition* and *change*. They defined Business Process reengineering[16] (BPR) as:

> "The fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed." [Hammer and Champy, 1994]

While Hammer backed off a bit on the need for these changes to be "radical" [Hammer, 1996], he did reinforce and detail the most significant aspect: process. The notion of a "process centered" organisation is one which executes a number of critical introspective steps ([Hammer, 1996], pp. 13–17):

[16] See also [Davenport, 1993] for an authoritative treatment of BPR.

- Recognise and name organisational processes
- Ensure **everyone** is aware of these processes and their role in them
- Measure and communicate process performance
- Actively manage the processes

Understanding, reengineering, and managing organisational processes requires a great deal of effort and investment. The payoff for these investments can be quite high, but the complexities of these tasks are such that they can easily lead to failure (cf. BPR lessons at CIGNA [Caron *et al.*, 1994]). One of the reasons for these failures is the inability to see the downstream impact of proposed changes. Processes often interact in detailed ways that make it difficult to fully understand the overall effect of change. This is underscored by the perspective that "an understanding and appreciation of the constraints on the process can ... only be achieved through a holistic or systemic view" [Alderman, 1997]. A range of BPR tools and methods have been designed for providing insight into the process gestalt.

2.12.1 BPR Tools/Methods

A number of commercial and research-based tools have been produced for BPR tasks (e.g. BPWin, Optima!, IDEF3-based ProCap/ProSim, oCTAVe, ProcessLink, Ithink, etc.). Both the ESRC-funded Business Processes Resource Centre[17] at the University of Warwick and the Business Process reengineering Advisory Group at the University of Toronto's Enterprise Integration Laboratory have created repositories [Gruninger, 1996] of references to such tools. The latter have also produced a set of properties for characterising BPR tools [BPRAG, 1995]:

- Integrated enterprise models
- Analysis (problem-solving capability)
- Software functionality

[17] The University of Warwick's Business Process Resource Centre (BPRC) is available at: http://bprc.warwick.ac.uk/

  - Integration of enterprise models and tools
  - Model management
  - New ways of building models
  - Project management tools

- Visualization and Communication
- Intended Users

Unfortunately though this characterisation appears to blur the distinction that other researchers have made between tools and methods for: *Business Process Analysis*, *Business Process Modelling* and *Business Process reengineering* [Alderman, 1997]. For example, IBM's Business System Development Method (BSDM) [IBM Corp., 1992] and tools which support it (cf. [Chen-Burger and Robertson, 1998]), appear to address aspects most closely related to analysis and modelling whereas tools such as oCTAVe provide support for streamlining or "radically" modifying process definitions (e.g. by identifying non-value adding steps).

Rich AI-based plan representations have been proposed to "support the modelling, analysis, and reengineering" of these processes [Tate, 1993b]. Tate lists the potential support for these activities:

- reliable capture and maintenance of process knowledge and models
- making decisions based on knowledge based simulation and analysis
- synthesis of plans and schedules
- reengineering parts of a process or plan
- reliable execution of processes and plans
- simulation, animation, and explanation of processes and plans

The focus is on open plan representations that can be easily inspected and manipulated. With these enriched representations the process team can operate at higher levels of analysis and can utilise the tactical planning strengths of an automated planning system to synthesise processes, highlight conflicts, etc. (see the PLANIT work we cited in Section 2.3.5). This knowledge must be communicated effectively and efficiently alongside the other tools used in the process reengineering setting. In particular, a core issue which needs to be addressed is the development of models which represent the essentials of business processes and decision models, enabling unambiguous analysis, visualisation, planning and reporting [Drabble and Beck, 1995].

2.12.2 Knowledge Management

Knowledge management has become a central concern for many large-scale organisations [Davis and Botkin, 1994, Davenport *et al.*, 1996, Quinn *et al.*, 1996, Nonaka, 1994, Chan Kim and Mauborgne, 1997]. Knowledge management is, in some ways, on the other end of the spectrum from process reengineering as an approach toward improving organisations. Process reengineering is focused on drastic process changes that produce significant cost reductions, whereas knowledge management focuses on effective knowledge creation and the use of available knowledge. Part of what knowledge management addresses is the specification of what actions are necessary to achieve better usability and added value [van der Spek and de Hoog, 1996]. This may include:

- Planning the actions to use knowledge assets
- Determining how to enact actions
- Monitoring those actions

These tasks are very similar to the ones that planning researchers have focused on in developing plan representations (e.g. representing activities, constraints on those activities, associating resources, time commitments, etc.). It is possible that enriched plan representations can be put to use in these settings by helping to define intelligent ways to enact these processes and automatically "flow" data, or work products, through their organisational processes. In fact, planning research has been used as representational input for an international body that is concerned with the development and promotion of workflow standards [Hollingsworth, 1994]. This body, the Workflow Management Coalition (WfMC), defines workflow as:

> "The automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules."

Managing workflow knowledge, like managing AI plans, requires reasoning about activities' effects and conditions, resource requirements, and the application of constraints at various levels. Organisations interested in modelling and/or generating workflow options require a representation similar to those described above in practical planning scenarios.

2.13 Connecting and Extending Past Research

> *The execution of this thesis work has been carried out in association with the O-Plan, Open Planning Architecture team. Part of the emphasis of this research has been to build on the past and current efforts associated with this project. Throughout the literature review, we have cited some of these efforts. In this section we will briefly recap the relevant work and outline some of the possible research opportunities.*

The work on the O-Plan project [Currie and Tate, 1991, Tate *et al.*, 1994c, Tate *et al.*, 1996, Tate *et al.*, 1998a] has involved several threads of research over its 16 year life-span. Many of the threads have evolved over time to respond to varying shifts of research emphasis. Opportunities exist to revisit and extend prior O-Plan work in addition to looking for ways to connect separate threads which would combine the strengths of various O-Plan related concepts. The following research opportunities are just some of those we highlighted toward the beginning of this dissertation work:

1. An <I-N-OVA> -Based Interlingua: Perhaps one of the most significant opportunities involves work with the <I-N-OVA> Constraint Model of Activity (see Section 2.3.4.3). While <I-N-OVA> can be seen as a model which underlies languages such as Task Formalism it is possible that a more direct use of its terms

and concepts could feed into the development of a language for expressing rich plan/process knowledge. This foundation might then be used as an interlingua (see Section 2.3.4.2) between various planning tools.

2. Human-Computer Interaction (HCI): As we showed in Sections 2.3.6.1 and 2.3.6.2, earlier work had been performed on the Task Formalism Workstation which provided a level of planning Human-Computer Interaction. Much of the technology for creating HCI applications has evolved since this early effort. As we showed in Section 2.3.3, the demand for including a human "in the loop" is on the increase. How would these tools relate to the <I-N-OVA> language discussed in 1? What type of tools are required?

3. Decision Support: Past O-Plan research has referenced the possible use of design rationale (see Section 2.5) for communicating information about plan decisions (see Section 2.3.2). In fact, even earlier than this was the related Nonlin work on utilising decision graphs (see page 35). How would this rationale be managed and presented in a tool (2) and how would this knowledge relate to the <I-N-OVA> language (1)?

4. Requirements Analysis: We cited earlier O-Plan work on utilising the CORE requirements engineering methodology to aide in eliciting requirements for a planning domain (see Sections 2.3.6.1, 2.3.7.3, 2.11). Unfortunately there wasn't enough detail on this early work on how this might be used. Also, how would it relate to the questions of tool support (2) and knowledge representation (1) raised here?

5. Organisational Process Management: Finally, as we showed in Section 2.3.5 O-Plan ideas and representations have been used in helping organisations manage knowledge about their processes. This was also explored in Section 2.12 in which we outlined the possible use of AI planning in the modelling, analysis, and reengineering of organisational process knowledge.

2.14 Conclusion

The research opportunities/issues which we outlined in Section 2.13 formed part of the basis for this thesis work. As we can see from our model of the space of fields and areas which are relevant (Figure 2.1), several elements implied by these research opportunities have been addressed in one way or another across a range of disciplines. They each offer possible alternative approaches to extending and connecting these planning research threads.

For example, issue 1 might be addressed with techniques from knowledge sharing (Section 2.7) and logic-based representations (Section 2.4). Several aspects related to issue 2 were discussed in mixed-initiative approaches (Section 2.3.3) and in existing AI-based planning toolsets (Section 2.3.6). A possible approach to incorporating plan decisions via design rationale (Section 2.5), as we indicated in issue 3, was outlined on page 36. Much has changed in requirements engineering (Section 2.11) and knowledge acquisition (Section 2.3.7) since the early work on CORE which was listed under issue 4. Finally, it was envisioned that some kind of unification of the work on these opportunities would go some ways toward providing an *integration framework for managing rich organisational process knowledge*.

Chapter 3

Methodology and Design of CPF

In Section 1.2.3 we introduced our Common Process Framework (CPF) solution to the problem of synthesising and managing organisational process knowledge. We outlined the various components which provide representations, methods and tool support. In the preceding literature review we toured research which has targeted various aspects of this problem. This chapter provides a detailed presentation of CPF and shows how our work has incorporated various approaches and fused them with our central focus of applying AI planning and plan representations.

3.1 Introduction

We begin our presentation of CPF by introducing the abstract design and discussing some of our design methodology. This design will show how the components interrelate to provide a unified integration approach. We will discuss the justification of this design vis-a-vis our research hypotheses (see Section 1.2.2) and we relate this to the overall justification of our work (see Section 1.3). Finally, we provide a detailed overview of this chapter before we begin to examine the individual facets of the framework.

Our approach to the problem outlined in Section 1.2 lies in the definition of a Common Process Framework (CPF). As we said in Section 1.2.3 this framework is a structure for building organisational processes and world descriptions as well as being a structure for containing shared representations of this knowledge. The framework is *common* in that it cuts across industries, but it is also *common* to a range of people, platforms and applications which participate in the management lifecycle of this knowledge.

We will consider the design of CPF from two different perspectives. First we present a look at some of the high-level phases, the tools and methods which support them, and the work products produced. Then we will examine the CPF architecture in more detail considering the relationships between the various components.

3.1.1 CPF Phases

> *In this section we outline the distinct CPF phases and briefly explain the role of the tools and methods. Our goal is to illustrate a typical sequence of CPF events and to identify and motivate the development of intermediate CPF artifacts.*

In Figure 3.1 we outline a model of the CPF phases. As we can see there are two initial phases identified prior to the central activities of managing and synthesising process knowledge.

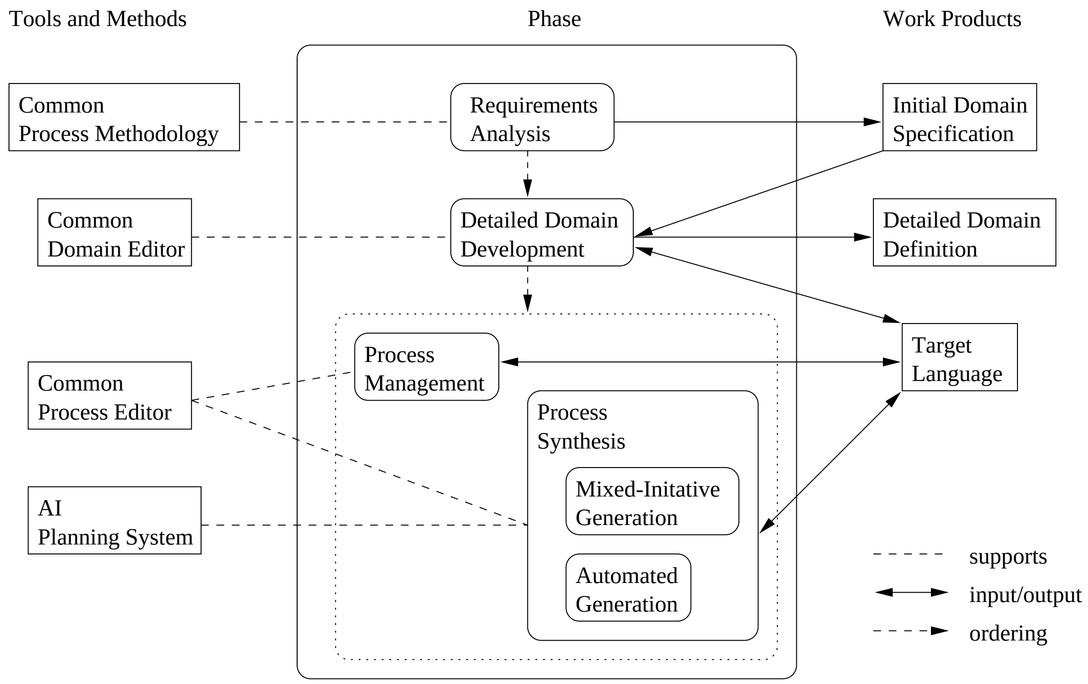

Figure 3.1: CPF phases, tools and work products

One of the major steps forward for this work, which is shared with others such as [Aylett and Jones, 1996] and [McCluskey and Porteous, 1997] (see Sections 2.3.7.3 and 2.3.7.5) is the identification of a *requirements analysis* phase prior to *detailed domain development*. The goal of the analysis phase is to both elicit and analyse knowledge of the domain or world description. The result of this phase is an *initial domain specification*. The specification is a more abstract artifact than the domain definition which is the result of the subsequent step.

This is analogous to the process of defining components in a distributed object-oriented information systems project (see Section 2.6). For example, when using CORBA the analysts first define the abstract interfaces for the objects using IDL (cf. [Madni and Mi, 1997]). IDL defines a strong separation between the specification of an object and the implementation of that object. Once the IDL has been defined, the developers add the details, or implementation of the object interface specification.

Thus, as we shall see in Section 3.2, the specification of a domain provides knowledge of the abstract interfaces between various *logical perspectives* in the domain. The methods for producing this specification are defined in the Common Process Methodology (CPM). This methodology is supported by the CPM toolset which we will discuss in Section 3.2.4.

Given the initial requirements specification, the detailed domain development phase is oriented toward refining this world description knowledge and creating the detailed domain definition. This involves making concrete decisions about aspects such as modelling levels and detailed constructs (e.g. resource constraints, variable constraints). Once sufficient detail has been added, the world description may be translated to a specific *target language*. For example, the target language would be Task Formalism (TF) if the intention was to use this knowledge within the O-Plan planning system. The main tool used to perform these tasks is the Common Domain Editor (CDE) which we will address in Section 5.2.

The dotted envelope in Figure 3.1 denotes a set of phases. In addition to the phases of process synthesis and process management which we have identified, there may be other phases included such as simulation, or evaluation phases. In a process synthesis phase, we may enlist the assistance of an AI planner, such as O-Plan or SIPE which utilises the target language domain work product produced in detailed domain development. This might involve fully automated or mixed-initiative process generation. The generated process may be visualised during mixed-initiative generation or edited during subsequent process management steps with the Common Process Editor (CPE). We examine the role of the CPE in Section 5.3.

3.1.2 CPF Architecture

The examination of the CPF phases in Section 3.1.1 introduced the high-level structure and component relationships. In this section we delve into CPF's architecture in more detail in order to describe some of our design choices and general approaches to integrating process knowledge.

The design methodology for CPF is centred around the AI planning model (see Section 1.1), a design-based perspective of process knowledge (cf. [Tate, 1996d]), and a knowledge-sharing approach to integration (see Section 2.7). The result of applying this methodology is the CPF architecture illustrated in Figure 3.2.

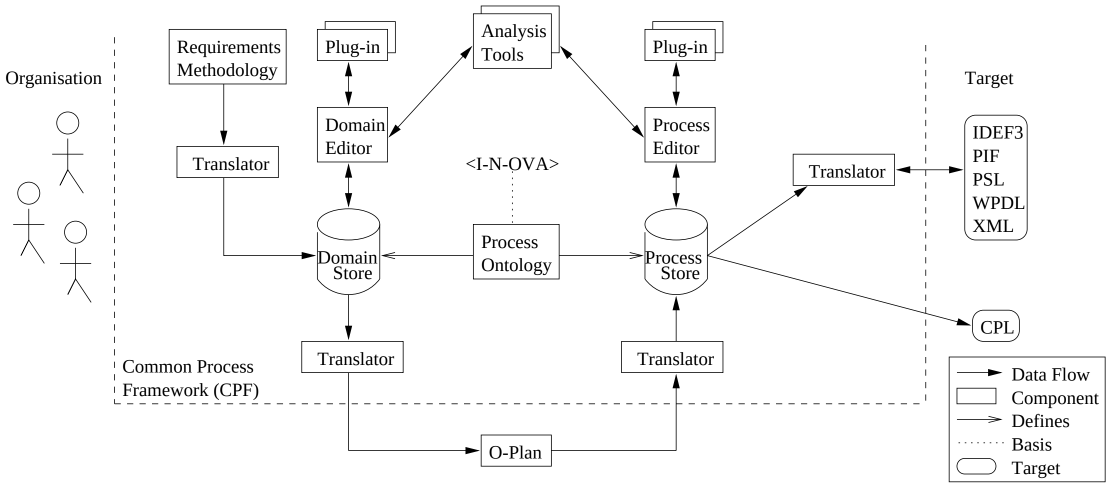

Figure 3.2: Generic CPF architecture

On the left hand side of Figure 3.2 we indicate an organisation. This organisation has some agents who may hold purposes or perform behaviour. On the right hand side we indicate a set of "targets" which an organisation might have for its process knowledge. These targets might be specific languages for expressing process knowledge, such as IDEF3 [Mayer *et al.*, 1992] or the Workflow Process Definition Language (WPDL) [WfMC, 1994]. Standing between these two is the architecture of CPF. We now walk through the typical CPF phases (Figure 3.1) again, but with a bit more focus on the details.

Before doing so, we recall back to the four areas of process management support which were cited on page 2. Throughout our overview of these architectural components, we will discuss how the components, both in whole and in part, serve to provide examples of the *user communication*, *formal analysis*, *systems integration* and *knowledge acquisition* axes.

Let's begin with the support for *knowledge acquisition*. An organisation, faced with some implicit knowledge of their overall organisational processes on the one hand, and with the capability of tools such as an AI planner on the other hand, must somehow bridge the gap. The first CPF component which comes into play is the requirements methodology. As we shall see, the methodology guides an organisation through the development of a series of intermediate representations. The structure of the representations helps an organisation to focus on the acquisition of various aspects of the domain or world description knowledge. As we mentioned in Section 3.1.1 this results in the development of an initial domain specification.

An obvious question then is "how is the initial domain specification expressed?". In order to answer this, we must bring in another axis, *systems integration*. Our approach to systems integration is based on knowledge-sharing (see Section 2.7). The representations used within CPF are ontologically underpinned. This is shown in the centre of Figure 3.2. As we can see, a process ontology defines elements in both a domain store and a process store. This process ontology (CPO) is based on the <I-N-OVA> constraint model of activity [Tate, 1995, Tate, 1996c]. The actual expression of world and plan description instances in the knowledge stores is via a common process language (CPL). Our systems integration support includes the notion of translation via Common Process Translators (CPTs), into and out of CPL. We cover CPO, CPL, and CPTs in Sections 3.3.1, 3.3.2 and 5.5 respectively.

Thus the initial domain specification will be expressed within the domain knowledge store following a translation step. This translation maps the largely graphical notations from the requirements methodology into the elements of the process ontology (i.e. lexical elements of the CPL). This initial specification may then be directly accessed by the domain editor (CDE). The task of the domain editor sessions is to provide the information which will transform this initial world description into a detailed domain definition. Thus we have also touched on a third axis of process management support, *user communication*. In effect this communication involved user-to-user communication, viz. one or more requirements analysis agents communicating with one or more domain editing agents. This communication was facilitated by the translation and visualisation support provided by CPF components.

The domain editing process can be specialised with various plug-ins. One of the plug-in examples we developed allows a family of constraint expression support modules to be accessed and utilised at runtime. We will describe this approach in Section 5.2. In addition to the plug-ins, we illustrate connectivity between the domain editor and other stand-alone tools. As an example of this, we developed the common process assistant (CPA). The CPA and its connection to CDE is similar to the expert system and "expert system hook" which was envisioned for the TF Workstation (see Section 2.3.6.1). The CPA provides us with an example of the final process management support axis, *formal analysis*. As we will show in Section 5.4, process knowledge may be sent to the CPA for a knowledge-based analysis. Our currently implemented analysis detects problems with temporal relationships and provides advice for addressing these issues.

At this point in our trace of the flow of events through the CPF architecture, we have arrived at a completed detailed domain definition. It is possible then that we would like to communicate this knowledge to tools which are external to the CPF. For example, we may be interested in synthesising a particular organisational process based on our knowledge of the domain. As opposed to the requirements translation where we were mapping into CPL, we are now interested in mapping out of CPL. Specifically, we would like to map the domain knowledge into an AI planning domain language to enlist the generative capabilities of an AI planner. The result of the AI planner is a newly synthesised organisational process.

This again involves a translation of this external process knowledge back into the shared interlingua of CPF. As we did with the world description, this newly synthesised plan description may now be further edited and managed. This time we are using the common process editor (CPE). CPE and CDE share a common presentation, but differ in ways which we will address in Sections 5.2 and 5.3. As with CDE, CPE has access to various plug-ins and external analysis tools.

An individual organisational process might then be translated to a specific process modelling language or representation (e.g. IDEF3). This would enable the use of the process knowledge in tools which are designed for this representation (e.g. using ProCap for an IDEF3 representation). Alternatively, this knowledge might be exchanged in what has been termed a hub-to-hub exchange in the ontological community. A hub-and-spoke **model** is outlined in Figure 3.3. This can be considered to be a higher-level projection of Figure 3.2.

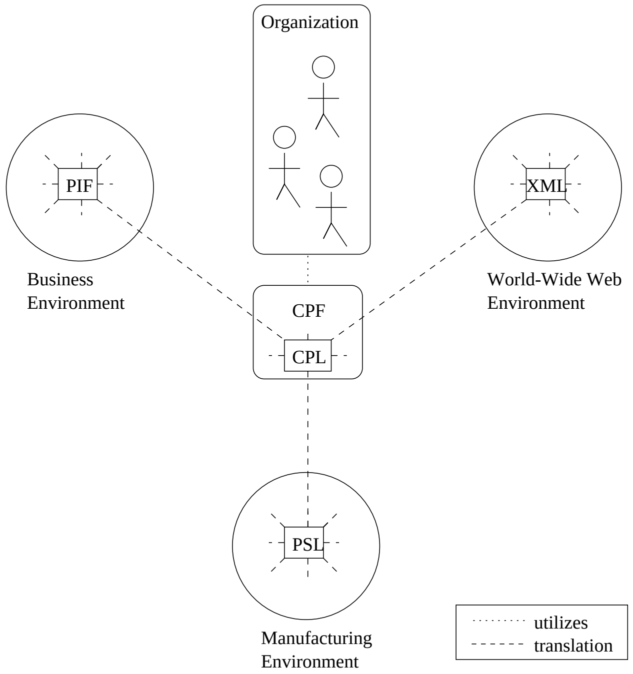

Figure 3.3: Hub-to-hub sharing across environments

In a hub-and-spoke model, various environments are identified. Within each environment a “hub” exists which denotes some shared interlingua for that environment. Knowledge-sharing within an environment (i.e. translations between the hub and other representations within the environment) is indicated by its “spokes” which are attached to the hub. Some spokes are said to be “hub-to-hub” because they involve an exchange across environments. In Figure 3.3 we illustrate three such exchanges between CPF's hub and hubs in a business, manufacturing, and internet or World Wide Web-based environment. An organisation might first utilise the common process framework to develop and edit its process knowledge. As we can see, the hub of CPF is the common process language. This knowledge can then be translated to a hub in another environment to support environmental interoperability. For example, the PIF interlingua supports exchange in a business environment [Polyak, 1998e, Polyak *et al.*, 1998], PSL for a manufacturing environment [Polyak and Aitken, 1998] or XML within the WWW [Lassila, 1998].

This concludes our overview of the CPF architecture. To sum up, we have discussed a framework which can be used to address the four areas of process management support: knowledge acquisition, user communication, formal analysis, and systems integration. We have described how the tools work together to integrate and improve the methodology of synthesising and managing organisational process knowledge. Throughout this chapter, and in Chapters 4 and 5, we will bring our focus onto the individual components we have identified in order to provide the details required to thoroughly explain our approach.

3.1.3 Justification

> *In Section 1.3 we introduced some of the justifications for our work in this research area. After introducing more detail on the design of our approach, we will now reflect on justifications of this design before moving forward with a detailed review of the framework components.*

Throughout Sections 3.1.1 and 3.1.2 we have discussed the high-level design of our solution to the research problem. As we have shown, this solution spans a lifecycle process for managing knowledge of agents, their purposes, and their behaviour. One question which we may ask is why we are interested in this lifecycle process as opposed to focusing this work on just an individual task within this process. For example, we may have chosen to simply work on the process ontology, the interchange language, or the application of a requirements methodology. The justification for this scope is related to Hammer's understanding of a business process. He states

> "The difference between task and process is the difference between part and whole. ... Only when [the tasks] are all put together do the individual work activities create value." [Hammer, 1996, pp. 5]

Indeed, our work is on understanding and providing knowledge of at least one way in which the lifecycle process involved in applying AI planning to organisational process management could be deployed. Given this knowledge of how the overall process provides value, we may later introduce improved tasks or products to increase this value (e.g. better methodology or tools, improvements in translation technology, etc.)

Another criticism we may have is of the appropriateness of AI planning for this research problem in the first place. Throughout the literature review we have pointed to various projects which have successfully applied research from this field to the problems of managing process knowledge (see Section 2.3.5). In fact, one of the American Association for Artificial Intelligence (AAAI) workshops is currently dedicated (July 1999) to "intelligent software engineering" given the following observed trends

- Knowledge representation methods provide efficient means for expressing business knowledge.
- Automated tools are required to reflect over business knowledge to identify what is missing or could be effectively changed.
- Knowledge base verification techniques can critique the structure of a knowledge base/specification.
- AI researchers now realise that software engineering provides the best testbed for AI tools and techniques.

If we accept that AI planning and plan representations are appropriate, then we are faced with the question of justifying our selection of <I-N-OVA> as the basis for our approach. As we described in Section 1.4, we reviewed several other alternatives [Polyak and Tate, 1997] in conjunction with our work on the Process Specification Language and reported on our results which showed that <I-N-OVA> had the flexibility and comprehensive approach to meet these needs.

One category which we haven't discussed in much detail yet is our set of implementation decisions. One of these technical decisions that we address in this justification section and which we will discuss more in Chapter 5 is our approach to common platforms. We believe that the CPF components should also provide *common* access across platforms rather than being tied to any one operating system or location. For example, we have written CPE and CDE as Java applets so that these tools can be downloaded and run on any machine possessing a Java Virtual Machine. The methodology (CPM) toolset has been created in a multi-platform case-tool (see Section 3.2.4). Standard protocols such as FTP and TCP/IP are built into the tools to support data exchange between physical locations.

Looking back at our original hypotheses in Section 1.2.2 we can justify our design as a method for testing these assertions. H1 through H3 have to do in part with the epistemological adequacy of the representation used in the domain and process knowledge stores. This serves to validate the process ontology and the underlying <I-N-OVA> model. Hypotheses H4, H5 and H10 involve effective translation and sharing of this knowledge as we described in Section 3.1.2. H6 deals with a specific category of knowledge for which we introduced a classification in Section 2.3.2. We will look at this issue of rationale in more detail in Chapter 4. H7 describes an important challenge to the expression of this knowledge (i.e. Common Process Language) which we discuss in Section 3.3.2. Finally, H8 has been explored by our work on CPE and H9 spans the work on CPM and CDE.

We can link these justifications back to those presented in Section 1.3. This design represents work which has been relatively neglected in the AI planning literature. While there have been applied projects which have looked into various aspects of the problem, there is a lack of structured options or designs such as this for those seeking to utilise the AI planning model. The importance of this area can be viewed through the usefulness of its potential application. Organisations can benefit from an understanding of how to incorporate or integrate the capabilities and representations from AI planning via this approach. We can also see that this design revisits past work on the TF Workstation and CORE and seeks to join it with <I-N-OVA> and other current research threads.

3.1.4 Overview

For the remainder of this chapter and extending into Chapters 4 and 5, we will examine each of the framework components. Throughout this examination we will provide examples from the house building domain which we introduced in Section 1.2.1 and which we will revisit in more depth in Chapter 6. We begin by presenting our incorporation of a requirements engineering methodology. Following that we discuss the shared representation which entails both the ontology definition and the interlingua as well as a mechanism for extension. This provides us with core elements to express a space of behaviour. We address the concept of incorporating the design rationale along with the design of a process in Chapter 4 before moving into a presentation of the CPF toolset. In Chapter 5 we will detail the domain and process editors, the knowledge-based process assistant, and the set of translators. Chapter 5 also brings together the spaces of behaviour and decisions in the implementation of the tools.

3.2 Requirements Methodology

> *We begin our examination of the CPF components with the process requirements methodology. The goal of the methodology is to provide a more disciplined approach to producing domain descriptions in applied organisational settings which encompasses requirements capture, analysis, and specification.*

Process engineering involves a search for new models of organising work. This activity cuts across industries. For example, manufacturing companies formulate steps for building a new product and suppliers define activities which are executed in the enactment of material and product supply chains. In either case, the amount of detail required to adequately describe these steps or activities can be difficult and time-consuming to manage. Interactions between planned steps exacerbate this process and complicate the analysis of proposed manufacturing or business plans. In this thesis, we argue that AI planning representations and planning systems can offer support for these activities. In fact, in Section 2.12.1 we cited ways in which AI planning research may specifically provide benefits. One of those ways involved reliable capture and maintenance of process knowledge and models.

Practical planners require a "knowledge rich" domain model that allows them to integrate efficiently given the demands of the surrounding environment. In order to transfer this research in AI planning to practical organisational process synthesis, we need to provide support for building this domain model. A domain model is constructed to provide information about the activities that may be performed (at various levels of abstraction) as well as the tasks which may be proposed to the planning system. Unfortunately, as we have cited, acquiring and maintaining this domain knowledge is currently considered to be a highly significant bottleneck in utilising planning systems [Wang, 1996]. The activities involved in discovering, engineering, documenting, and maintaining a set of domain constructs for most domain independent AI planning-based projects can be considered ad hoc and disorganised, at best.

3.2.1 Prior Edinburgh-Based Research

In this section, we introduce the particular past research thread which we were interested in revisiting and extending as part of our thesis work. For a review of related approaches, see Section 2.3.7.3.

Domain capture and modelling has been an issue in Edinburgh-based planning research as early as the work on the Nonlin [Tate, 1977] planner. The original O-Plan overall architecture and system design, which dates from 1983, outlined a need for a defined methodology which would guide users performing various roles in the acquisition and analysis of domain requirements for planning [Currie and Tate, 1991]. This planning lifecycle methodology was envisioned as encompassing a set of standardised activities and methods which had well-defined design criteria, techniques, and tools. As we have said earlier, this was proposed to assist in transforming planning domain development from a craft towards more of an engineering activity.

The domain description language used by both the Nonlin and O-Plan planners is the Task Formalism (TF) [Tate, 1977, Tate *et al.*, 1994a]. Early prototyping efforts on a PERQ-based TF Workstation [Tate and Currie, 1984, Tate and Currie, 1985] demonstrated tool-support for the domain modellers (an expert providing the structure of the domain and specialists providing the details) and planners (acting in any one of a range of roles). This tool was designed to include an "intelligent assistant" which would interact with the user via a structured dialogue which was tied to a specific domain development methodology. Research was conducted into a requirements engineering methodology which could be adapted for use in this way. The Controlled Requirements Expression (CORE) [Mullery, 1979, Curwen, 1991] was proposed for structuring these domain management activities. Unfortunately this work had been set aside following the initial exploration of these ideas [Wilson, 1984]. Recently, we provided a review of this past work which pointed toward the development of a new methodology which could revitalise this area of research [Tate *et al.*, 1998b, Tate *et al.*, 1999b].

3.2.2 CORE

> *As part of the CPF architecture, we have picked up on these earlier research ideas and adapted and incorporated work on the CORE requirements methodology. In this section, we present an overview of CORE before moving on to describe the methods and work products.*

COntrolled Requirements Expression (CORE) was a method developed by British Aerospace (Warton) and systems designers in the late 70's [Mullery, 1979]. As was mentioned above, initial work at Edinburgh sought to relate this method to engineering AI planning domains. Over time, the method has evolved and CORE now provides techniques for requirements capture, analysis and specification [Curwen, 1991]. The method can be used to partition problems into manageable modules which can be assessed using CORE analytical techniques. This helps to ensure that the requirements for a specification are complete and consistent. Some of the strengths of this methodology include decomposability of requirements and traceability mechanisms between different levels of requirements.

The CORE specifications are expressed in terms of graphics, structured text and specialised notations. These resultant requirements models start from operational requirements which influence functional requirements and, in turn, impact implementation requirements (with non-functional requirements acting as functional and implementation constraints). Viewpoints are used as logical partitionings of the system under consideration. These are divided into **bounding viewpoints**, which may be viewed from a planning context as providers of unsupervised conditions and **defining viewpoints** which are analogous to activities which can achieve conditions. Viewpoint decompositions correspond to node expansions. The CORE notion of "scope" addresses choices between elements which may be included in the domain, and breaks them down into "local scopes" which designate responsibilities for domain specialists.

An adaptation of the CORE methods can be used to structure the activities of users acting in particular roles throughout the engineering of a domain. For example, a domain expert divides a domain into a series of tasks to be completed by specialists. Domain specialists can list the assumptions they will be making within their scope (e.g. for a supply chain domain these assumptions may include: order validated, delivery notice sent, etc.). Specialists can retrieve previous parts of a domain specification to modify. For each specification, a viewpoint decomposition process is applied to it. This includes some model checking based on CORE analysis techniques.

CORE provides specialised techniques for inspecting the evolving specification. One example is the "viewpoint to viewpoint role-playing" technique. Using this approach, structured documents are produced which define a particular perspective within the domain (e.g. for a supply chain domain this may be between a retailer and a distributor, or between a manufacturer and a transportation company, etc.) Techniques such as this one aid in combining the viewpoints by showing where conflicting requirements are present.

3.2.3 Workproducts and Methods

> *In Section 3.2.2, we presented an overview of CORE and discussed some of our perspectives on its application to engineering AI planning domains. We will now present the specific methods and work products associated with our adaptation of its enactment. We will refer to this adaptation as the Common Process Methodology (CPM). This section is based on our description of CPM which we provided in [Polyak, 1999].*

The adaptation of the CORE methods and workproducts for use in the CPF is called the Common Process Methodology (CPM). CPM's aim is to partition the domain into manageable modules that can be analysed using rule-based and human-supported interaction techniques. This is strongly based on the way CORE partitions a software system design. As in CORE, the intermediate specifications will be expressed in terms of graphics, structured text, and specialised notations. The focus is on decomposing requirements into further detail and providing traceability mechanisms between different levels of requirements.

Some of the essential attributes of CPM requirements include

Modularity The specification of a domain can become increasingly complex and it is important to break it down into modules which will enable it to be analysed. This will also assist domain specialists and experts in assessing the impact of various modifications to the specification.

Hierarchical Many AI planners support a HTN-style [Sacerdoti, 1975, Tate, 1977, Erol, 1995] of domain abstraction. This should be supported in the specification. Structured levels of increasing detail in the specification also make it easier to understand.

High Quality High quality is meant to indicate the aim toward a balanced set of competing properties which include requirements which are: unambiguous; consistent; complete and visible. Visibility in this context means easily accessible and readable for any specification user.

Verifiable Requirements need to be in a form such that they can be verified for consistency and completeness.

The CPM process of eliciting and engineering these requirements is composed of various structured activities which are illustrated in Figure 3.4. To reiterate from above, the goal of these activities is to elicit and partition the domain knowledge and to incrementally move toward an initial domain specification. In this section, we review each of these steps and discuss the work products produced at each stage.

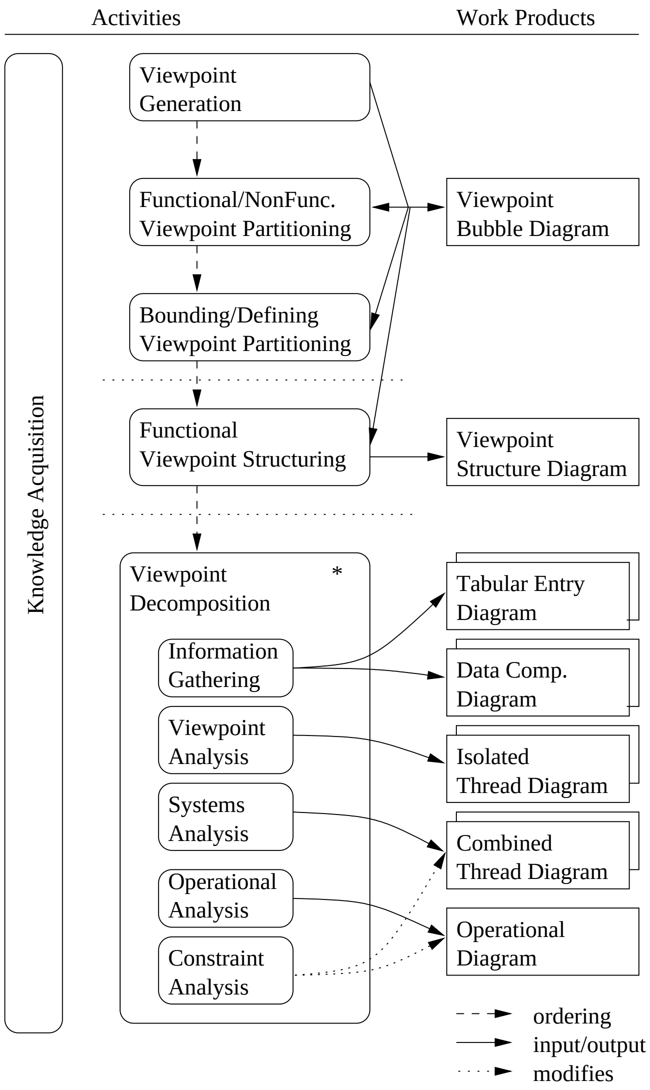

Figure 3.4: Common Process Methodology steps

3.2.3.1 Viewpoint Generation

The first step in CPM is *viewpoint generation*. CPM draws on the CORE notion of *viewpoints*[1]. In the subsequent steps, the methodology will provide guidance on how to modularise a domain into a hierarchical structure through the use of these abstract viewpoints. Viewpoints have been defined in CORE [Curwen, 1991] as

> "A viewpoint is a logical partitioning of the system under consideration."

[1] Research on viewpoints is ubiquitous in the requirements engineering field, cf. [Finkelstein *et al.*, 1994, Kotonya and Somerville, 1996, Easterbrook and Nuseibeh, 1996].

Viewpoints can be used to examine the domain in a variety of ways which enables the domain analyst to focus and capture information relevant to a specific perspective. In this way, viewpoints play the role of breaking the domain down into a number of modules. In viewpoint generation, the domain expert along with the various specialists conduct a session in which potential viewpoints are listed. This is similar to the "brainstorming session" suggested in the scoping phase of the ontology capture process described in [Uschold and Gruninger, 1996].

As suggested in Figure 3.4, this process is centred around the development of a Viewpoint Bubble Diagram (VBD). Each candidate viewpoint is simply drawn as another bubble on a diagram space. This type of initial knowledge acquisition task is typically found in many of the KBS-based methodologies (cf. the data conceptualisation stage in KEATS [Motta *et al.*, 1991], see also Section 2.8). Viewpoints may be proposed and rejected as an exploration of possible entities yields more knowledge about the scope of the domain.

In order to illustrate the result of this step, and of the subsequent steps, we will be referring to example work products developed for various domains including the three pigs domain (see Section 1.2.1). The "three pigs" work products are listed in Appendix D. For example, the diagram in Figure D.1 reflects an initial VBD for this domain. This diagram was produced using the CPM toolset which we will cover in Section 3.2.4.

In addition to the input from experts and specialists, there may also be additional sources of information for generating domain viewpoints. The collection of this knowledge is reflected in Figure 3.4 as part of the on-going knowledge acquisition activity which is enacted throughout the CPM process. These sources may include

- Existing documentation about the domain (structured or unstructured).
- Existing organisational policies and procedures which influence the domain.
- Documentation or knowledge of other domains which are similar or which reflect "best practice" processes (cf. [Malone *et al.*, 1993]).

3.2.3.2 Functional/Non-functional Viewpoint Partitioning

The next step in applying CPM to the engineering of a process domain is to perform an initial partitioning of the proposed viewpoints into those which are functional and those which are non-functional. Depending on the complexity of the domain and number of viewpoints generated, this may be performed immediately after the initial generation phase or over a series of subsequent sessions. In order to determine which type a particular viewpoint may be, the following definitions have been adapted from CORE:

Functional Viewpoint A logical partitioning of the domain under consideration into modules that are transformers of information.

Non-Functional Viewpoint A group of requirements that modify or constrain the functional requirements of the domain.

The "transformers of information" part of the functional viewpoint definition is very important. Functional viewpoints typically input data, transform it, and output the results. If a viewpoint cannot be characterised in this way, then it will be considered to be a non-functional viewpoint. Non-functional viewpoints constrain requirements. These constraints may be domain-wide or may only affect part of the specification.

For example, working with the VBD in Figure D.1 we can identify two probable non-functional viewpoints: security and cost. Both of these items reflect constraints on the possible functional aspects of the domain. The remaining viewpoints appear to be functional, i.e. transformers of information in the domain. As in CORE, CPM considers "function" to be the driving, master set of requirements. The identified non-functional requirements will be subsequently expressed either in structured text or in a specialised notation, whereas the functional requirements will play a central role over the next steps in the development process.

3.2.3.3 Bounding/Defining Viewpoint Partitioning

In this phase, the functional viewpoints are examined in more detail to derive another partitioning of this subset of proposed domain elements. We will consider these viewpoints to be one of two types: bounding or defining. The following definitions can be used to determine the type.

Bounding Viewpoint A bounding viewpoint is a functional viewpoint which represents an outermost point or bounding edge in a domain. A bounding viewpoint is therefore used to generate a view of how the domain looks from a particular, fixed outer vantage point. Bounding viewpoints are non-decomposable.

Defining Viewpoint A defining viewpoint represents part or all of a segment of the domain being explored and is therefore used to describe how the domain functions internally. Defining viewpoints may be decomposed.

As an example, in our house building (three pigs) domain example we can identify the following items as being "bounding viewpoints"

- Pig
- Construction company
- Material provider
- House Inspector

While we may uncover more bounding viewpoints as we continue to refine our knowledge of the domain, we are interested in trying to provide a relatively complete set of top level perspectives.

Turning our attention now to the defining viewpoints, it may be possible to identify several general viewpoints which can be used to further subgroup these elements. Sometimes these general viewpoints are already present in the generated set. Alternatively, we may need to add new generalisations to group related viewpoints. Figure D.2 shows our results from applying this method to the functional set in Figure D.1.

In Figure D.2, we can see that we have changed the model to reflect our partition of the defining viewpoints in the domain into those involved with building the house and checking the security. This uses a Venn-like diagram notation. Both were already identified, but in this step we decided that the "build house" viewpoint actually contained many of the other viewpoints. This was also the case with the "build walls" viewpoint which actually encompasses the "purchase materials" viewpoint. As we shall see in other domains, it is also often the case that some defining viewpoint appears in multiple generalisations (e.g. "receive order" and "ship product" are common to different viewpoints within a supply chain domain).

3.2.3.4 Functional Viewpoint Structuring

In this stage of engineering the process domain, we have a Viewpoint Bubble Diagram as input which was produced and refined during the prior steps. The next task is to convert this VBD into a Viewpoint Structure Diagram (VSD). The VSD will provide the framework to capture and analyse the detailed requirements for the domain. This structure consists of nodes which represent viewpoints linked in a hierarchy of detail with increasing detail towards the bottom of the diagram and decreasing detail toward the top. The top level node created simply represents the entire scope of the domain (e.g. "Three Pigs Domain"). Bounding viewpoints are then arranged with respect to their relevant levels alongside defining viewpoints. In CPM as in CORE, a general guideline is that no more than five functional viewpoints should be chosen for each decomposition of the structure. This helps to keep the design compact and expressed at an appropriate level of granularity for human inspection.

The converted VSD for the example domain is shown in Figure D.3. The top level node, "Three Pigs Domain" is decomposed into one defined viewpoint which represents a generic planning task, "build house task". The domain is considered to be bounded at the top level by the "Pig" bounding viewpoint. For some domains, it is also useful to add an "Environment" bounding viewpoint at this level for relating the domain processes to external (i.e. unmodelled) influences. The numbering of these diagrams is based on the IDEF3 [Mayer *et al.*, 1992] style of process numbering: Parent.Decomposition Number.Unique ID.

The additional grouping viewpoints identified in the VBD were considered to be logically part of this top level. For example, we can see our split between "building the house" and "checking the security" at the next level along with the introduction of another bounding element, the house inspector. It should be stressed that there is no single "right way" of decomposing a domain. The important aspect is that the viewpoint structure addresses the whole of the required domain and agrees with the perspectives gained during knowledge acquisition.

In the next series of steps, CPM provides methods for decomposing the viewpoints structured in the VSD. The aim of this phase is to incrementally build a specification at each branch of each level in the hierarchy. These specifications are analysed and then finally combined to produce a uniform domain specification. This process involves the creation of Tabular Entry Diagrams (TED) which define the interfaces for the various viewpoints. From a given TED, we will derive a series of isolated threads of activity. These isolated threads will then be combined to unite flows between viewpoints. An operational analysis phase will bring all of these combined threads together to outline the overall structure of the processes. A final phase will add in the remaining constraints required for the domain.

3.2.3.5 Information Gathering

In this first phase of decomposing viewpoints, we will create a Tabular Entry Diagram (TED) for each functional viewpoint in our VSD. A functional viewpoint is used as a mechanism for focusing the specialist or expert's attention on a small area of the domain. A newly created TED must be assigned: a unique reference, a title, and a reference to its parent. Next, a series of 5 columns are provided which can be used to record information relevant to the viewpoint. These items are

Actions Something that transforms information. Question to ask: What actions does the viewpoint provide?

Inputs Information about the world that is required by an action. Question to ask: For each action, what are it's inputs?

Outputs Information about the world which is produced by an action. Question to ask: For each action, what are it's outputs?

Source The corresponding Tabular Entry Diagram which provides an input. Question to ask: For each input, do I know a specific source for the information?

Destination The corresponding Tabular Entry Diagram which provides an output. Question to ask: For each output, do I know the destination(s) of the information?

Recalling back to Section 3.2.3.2, we characterised defining viewpoints as transformers of information. The actions in defining viewpoints are similar to the business processes described in [Hammer, 1996]. For example, Hammer states

> "We can think of a process as a black box that effects a transformation, taking in certain inputs and turning them into outputs of greater value. ... The essence of a process is its inputs and its outputs, what it starts with and what it ends with. Everything else is detail." [Hammer, 1996].

The focus of a TED is to elicit the "interface activities" for a particular viewpoint. These interface activities expose functionality which may interoperate with other activities in another viewpoint. A given viewpoint may offer alternative activities which can require similar inputs or similar outputs. For example, "transport products via truck" and "transport products via plane" both provide the similar output of transforming the location of the products. Node notes may be attached to any of the above items to provide details which help to clarify the interpretation of the item.

For example, the TED in Figure D.4 presents the detail of the "build house" viewpoint. The viewpoint encompasses three main activities or processes, select-material, construct-house, and check-requirements. The select-material activity takes a material-preference from the Pig viewpoint as an input and outputs information on which material was selected to other viewpoints.

It is important to note here that the source and destination categories are not required. This is a major step away from CORE and is an important observation used in CPM. One of CORE's assertions is that the specification should eventually be entirely statically connected (i.e. all inputs and outputs map to a specific set of sources and destinations). Lessons learned from representations used in AI planning suggest that most of these links should be "dynamic"[2]. This permits users to build "generic" viewpoints which may be utilised in a various parts of the specification. The source and destination columns can still be used to model "known" links between specific viewpoints, but do not reflect a limiting set of such possibilities.

[2] Work by Systems Designer, Ltd. and AIAI in the early 1980's explored the use of AI planning to make such dynamic connections of inputs and outputs in the Analyst Workbench for CORE [Stephens and Whitehead, 1984].

This particular representation enables a domain specialist, or domain expert to perform one of two possible techniques in capturing these interface activities. These techniques are characterised by role-playing. They include

Viewpoint-to-Viewpoint Role-Playing (VVRP) In VVRP, the analyst assumes the viewpoint of the particular TED and then selects another viewpoint at the same level. The analyst then considers the possible exchanges which may take place between the two viewpoints. For example, if it is a supply chain domain the TED may be a "Customer" and the target viewpoint may describe the "Replenish Inventory" process which is enacted at a retail store. The analyst may then reason that the Customer will provide an "order-goods" activity which will output "selected-goods" that is required for "Replenish Inventory". Likewise, the analyst may conclude that from a customer's viewpoint, an activity should be provided called "receive-goods" which will take "retail-goods" as output.

Isolated Viewpoint Role-Playing (IVRP) IVRP is similar to VVRP, but this is a less constraining activity. Again, the analyst assumes a viewpoint for a particular TED, but then simply hypothesises what activities will be provided and what inputs and outputs will be present.

Individual domain specialists may be tasked in parallel with developing an IVRP for their scope in the domain. These diagrams then serve as focal points for discussions aimed at understanding the assumptions being made and for ameliorating the differences. In practice, VVRP and IVRP can be used in combination where necessary.

Another aspect of this phase involves support for analysing the data (i.e. inputs and outputs) which are specified in the diagrams. Separate Data Composition Diagrams (DCD) can be attached to individual data entries. The graphical notation for this can be traced back to [Jackson, 1975]. An example of a data composition use for the three pigs domain might be to define that a "material" input or output could be straw, sticks, or bricks (mutually exclusive).

3.2.3.6 Viewpoint Analysis

The next stage in the viewpoint decomposition phase involves viewpoint analysis. Viewpoint analysis is concerned with analysing each action of each viewpoint in isolation from one another. The main goal of this analysis is to determine: what starts a viewpoint action and what stops a viewpoint action. This is done by creating separate, isolated threads for each action and classifying each data input as being either: event data, control data, or data containing information. The following definitions are used to make these decisions.

Event Data When the data is generated the receiving action must occur, i.e. event data is a trigger which starts actions. Event data has no value, the data is either there or it is not there.

Control Data When the data generated is used to select the operation of different actions. These actions will be mutually exclusive. The selection may be based on the state of the control data or on limits of the value of the data.

Data Containing Information When the data generated contains information which is required to be used by an action.

We also need to consider whether the data is critical or non-critical. Critical data is characterised by data which once generated must eventually be used and used only once, i.e. it is consumed when read and is not overwritten. Non-critical data on the other hand is volatile and can be overwritten. This gives rise to the following possible reasons why each isolated thread action starts:

- time (which may also involve iteration)
- event data
- critical control data
- critical data containing information
- a missing critical data input

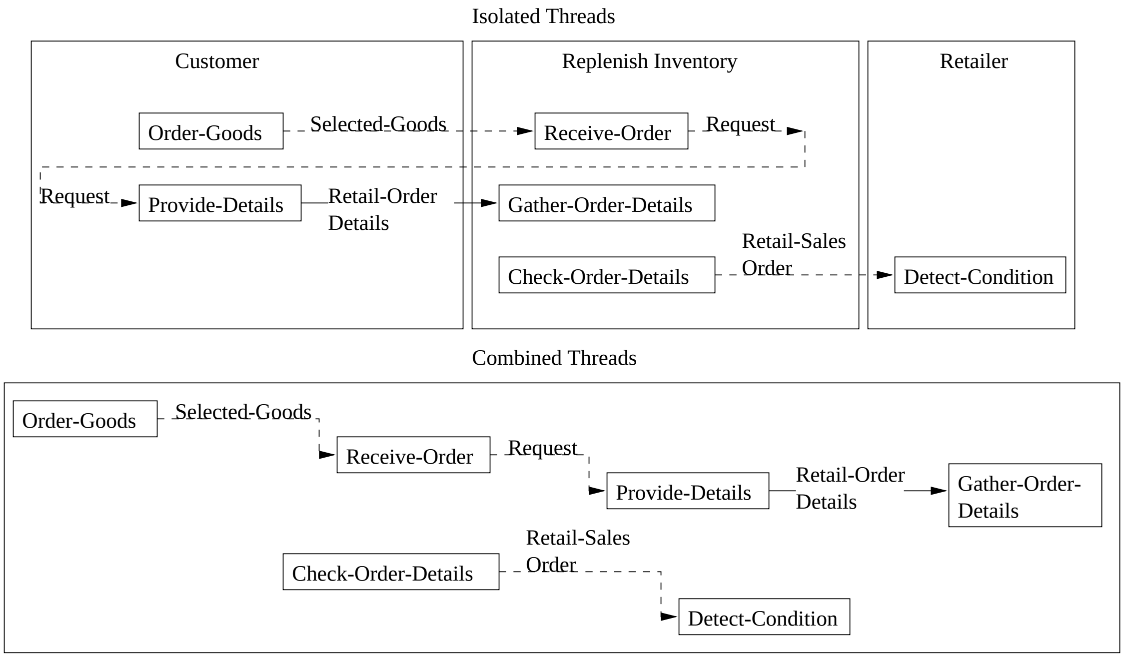

Figure 3.5: Creation of combined threads from isolated threads

The analysis will also determine the reason why each isolated thread action stops

- if it is started by time then it could only occur a specific number of times or over a range

- if it is started by time then it could be stopped by non-critical control data (e.g. enable/disable)

- if it is started by time then it could be stopped after an internal condition changes

- if it is started by critical input then it is a "one shot action"

The result of this stage is a set of isolated threads for each viewpoint. These threads provide a basic set of building blocks which can be used to create "combined threads". A combined thread can be thought of as an operator (in traditional STRIPS-style planning), schema (in Task Formalism) or a plot (in SIPE-2 [Wilkins, 1988, Myers and Wilkins, 1997]).

3.2.3.7 Systems Analysis

The analysis completed so far has been concerned with the modularisation of the requirements using viewpoints. For each viewpoint, actions have been analysed in isolation from each other further modularising the requirement. Systems analysis is the process in which the sequence and concurrency between actions at a given level is next determined. This requires a weaving of separate threads developed across viewpoints into a collection of combined thread diagrams. Each diagram represents a composite process, built from the basic isolated threads, which will be available for synthesising new "models of work".

In Figure 3.5, we can see how a sampling of isolated threads for a supply chain scenario may be combined. The customer, replenish inventory, and retailer viewpoints (at the top) were combined to form the two combined threads (at the bottom) which represent composite supply chain domain processes. The dashed lines represent critical data containing information (e.g. Selected-Goods) or critical event data (e.g. Request). The solid lines represent non-critical data containing information (e.g. Retail-Order-Details). The means of evaluating the relationships is to examine the nature (critical/non-critical) of the data crossing viewpoint boundaries. From the definition of critical data, it can be seen that an action produces critical data which then triggers the receiving action. This means that actions linked by critical data must occur in a sequence as the receiving action can only occur after the producing action.

The way non-critical data has been defined states that the generating action periodically produces new data that overwrites the previously determined value, the receiving action using the latest value available. In this case, the relationship between the producing and receiving actions is a concurrent one because the producing action does not have to occur before the receiving action. Thus, critical and non-critical data can be used as one means of detecting sequence and concurrency between the isolated threads of different viewpoints.

3.2.3.8 Operational Analysis

In systems analysis we produced a number of combined threads which show the required sequences (schemas) in the domain. Operational analysis is concerned with bringing all these separate combined threads together onto one diagram – the operational diagram.

The operational diagram represents sketches of potential lifecycles for high level processes. These high level processes may be considered to be "task schemas" in the Task Formalism. For example, in manufacturing, we may have a high-level "build product GT-350" (see Section 6.2.4) or in supply chain management we may have a "enact supply chain" process (see Section 6.2.2). The lifecycle is constructed by identifying the major events that occur within the domain for that high level task. In the manufacturing domain example (where the product GT-350 is a model car), this may include: making the interior, building the drive, adding the trim, making the engine, and affixing the drive.

The completed operational diagram, therefore, provides a total picture of how the domain will be logically configured for addressing a task. While building up this complete picture, a final check should be made to ensure that no actions which are required over the lifecycle are omitted. The resulting operational diagram effectively provides an index to the detail of the domain which is shown on the combined threads.

3.2.3.9 Constraints Analysis

The final technique in CPM takes us back to the non-functional requirements which we elicited back in the initial viewpoint generation phase. As mentioned previously, these non-functional requirements will introduce a modifying or constraining influence on the functional requirements developed above. They may involve adding new functionality or new flows to the combined thread or operational diagrams. These may also be expressed via detailed node notes as well.

To some degree, this sharp division between functional and non-functional domain requirement considerations is an idealisation. In practice, several constraining influences will have crept into the earlier analysis stages. However, this separate stage serves as a reminder to go back and consider those unaddressed aspects and to formally complete the work on the initial specification. For example, for the three pigs domain this requires analysing where the cost and security factors have been addressed in the domain.

3.2.3.10 Translation to CPL

At this stage, we have created a set of requirements diagrams which has provided documentation and gradually guided us towards an initial specification of the domain. The final step in this process involves translating our knowledge of the requirements into a language with which we can share it with other tools. Specifically, we are interested in utilising this knowledge as a starting point for refining it into a detailed world description. We will discuss this translation process alongside the other translators in Section 5.5.

3.2.4 CPM Toolset

The Common Process Methodology is supported by the CPM toolset. The toolset is a customisation of the HARDY[3] hypertext diagramming meta case-tool which was developed at the Artificial Intelligence Applications Institute (AIAI). The toolset runs on UNIX X Windows (Solaris 1.x or 2.x.) as well as Microsoft Windows (3.x, 95/98, NT).

The specialised diagram types (VBD, VSD, TED, etc.) have been encoded in HARDY in order to provide window-specific pallets and presentations. The pallets display the available diagram options (i.e. various node and arc types) from which a user selects. Arcs are constrained to only allow connections between appropriate nodes (e.g. an I-A arc on a TED can only relate inputs and actions, etc.). The support for hypertext linking and retrieval in the CPM toolset enables efficient, intuitive browsing between the various diagrams.

Besides the Human-Computer Interface benefits provided for creating and managing the work products, the CPM toolset also contains an embedded expert system shell for rule-based processing and analysis of the diagram contents. This is made possible by HARDY's built in link to NASA's rule-based and object-oriented language CLIPS 6.0 [Giarratano, 1994]. CLIPS close link with HARDY is utilised for preformatting newly created diagrams, analysing diagrams for completeness and consistency and exporting the initial specification for use within the Common Process Framework.

[3] See http://www.aiai.ed.ac.uk/project/ for information on the HARDY project at AIAI.

For this thesis work, the completeness and consistency checks we have encoded in CLIPS have to do with those described in CORE for the Tabular Entry Diagrams. These checks aid in detecting and correcting conflicts between different parts of the domain description. In the following section, we will present the basic language and axiomatisation which underlies the implemented CLIPS rules and representation of the diagram constructs.

3.2.4.1 Basic Language and Axioms for Tabular Entry Diagrams

In this section we summarise a simple axiomatisation of the tabular entry diagrams described earlier. We have sorts $\mathcal{A}$, $\mathcal{I}$, $\mathcal{O}$, $\mathcal{S}$, $\mathcal{D}$ for actions, inputs, outputs, sources, and destinations which may be entered on a Tabular Entry Diagram, $\mathcal{T}$. Variables are denoted by lower case letters (with or without subscripts), and constants are denoted by upper case letters (with or without subscripts). Unless otherwise stated, letters a, i, o, s, d, t (A, I, O, S, D, T) and used for variables (constants) of sorts $\mathcal{A}$, $\mathcal{I}$, $\mathcal{O}$, $\mathcal{S}$, $\mathcal{D}$, $\mathcal{T}$ respectively.

Firstly, we have the simple association that all objects of type $\mathcal{A}$, $\mathcal{I}$, $\mathcal{O}$, $\mathcal{S}$, $\mathcal{D}$ are required to be associated with only one T. So, we will repackage these all with appropriate 2-place predicates:

$$(\forall a)(\exists t).action(t,a) \tag{3.1}$$
$$(\forall i)(\exists t).input(t,i) \tag{3.2}$$
$$(\forall o)(\exists t).output(t,o) \tag{3.3}$$
$$(\forall s)(\exists t).source(t,s) \tag{3.4}$$
$$(\forall d)(\exists t).destination(t,d) \tag{3.5}$$

Thus, a given tabular entry diagram, T, contains a tuple of sets $< \{A_0,A_1,...A_n\}$, $\{I_0,I_1,...I_n\}$, $\{O_0,O_1,...O_n\}$,$\{S_0,S_1,...S_n\}$,$\{D_0,D_1,...D_n\} >$. Next we have the following relations which may be used to associate the objects within a particular T:

$$(\forall a,t,i).has-input(t,a,i) \supset action(t,a) \wedge input(t,i) \tag{3.6}$$
$$(\forall a,t,o).has-output(t,a,o) \supset action(t,a) \wedge output(t,o) \tag{3.7}$$
$$(\forall i,t,s).has-src(t,i,s) \supset input(t,i) \wedge source(t,s) \tag{3.8}$$
$$(\forall o,t,s).has-dest(t,o,d) \supset output(t,o) \wedge destination(t,d) \tag{3.9}$$

In addition, a T is required to have one and only one parent, T_{parent}. This hierarchy of parents ultimately stems from the top level node anchoring the VSD.

$$(\forall t)(\exists t_1).has - parent(t, t_1) \land \neg((\exists t_2).has - parent(t, t_2) \land t_1 \neq t_2) \tag{3.10}$$

From this hierarchy of parent relations we can infer a standard, transitive $ancest - of \subseteq \mathcal{T} \times \mathcal{T}$ relation which will be used in establishing the traceability of data interface definitions between modelled levels. This is inferred by the rules:

$$(\forall t1, t2).has - parent(t1, t2) \supset ancest - of(t2, t1) \tag{3.11}$$
$$(\forall t1, t2, t3).has - parent(t1, t2) \land has - parent(t2, t3) \supset ancest - of(t3, t1) \tag{3.12}$$

3.2.4.2 TED Specification Analysis

Given this representation, the following checks are made for the entire set of tabular entry diagrams. These checks can be considered to be of two types. Those checks which involve specification constructs within a single viewpoint and those which involve constructs which span viewpoints.

3.2.4.3 Within viewpoints

Within viewpoints, the following checks can be made in order to determine whether the requirements for that viewpoint diagram are consistent and complete.

Rule 1 All inputs have a possible source (either statically assigned or dynamically determined)

$$(\forall t, i).input(t, i) \supset ((\exists s).has - src(t, i, s) \land source(t, s)) \lor ((\exists t_1).output(t_1, o) \land i = o) \tag{3.13}$$

Rule 2 All outputs have a possible destination (either statically assigned or dynamically determined)

$$\begin{aligned}&(\forall t, o).output(t, o) \supset \\ &\quad ((\exists d).has-dest(t, d, s) \land destination(t, d)) \lor ((\exists t_1).input(t_1, i) \land o = i)\end{aligned} \tag{3.14}$$

Rule 3 All actions have at least one input or one output.

$$\begin{aligned}&(\forall t, a).action(t, a) \supset \\ &\quad (((\exists i).has-input(t, a, i) \land input(t, i)) \lor ((\exists o).has-output(t, a, o) \land output(t, o)))\end{aligned} \tag{3.15}$$

Rule 4 All items are correctly connected by lines.

See axioms 3.6 through 3.9.

3.2.4.4 Across Viewpoints

Across viewpoints, the following checks can be made in order to determine whether the requirements for the entire set of viewpoint diagrams are consistent and complete.

Rule 5 All inputs must appear as (direct/indirect) outputs on the given sources.

$$\begin{aligned}&(\forall t_1, i, s).input(t_1, i) \land has-src(t_1, i, s) \supset \\ &\quad (\exists t_2, o, d).t_2 = s \land output(t_2, o) \land i = o \land has-dest(t_2, o, d) \land \\ &\quad ((d = t_1) \lor (ancest-of(d, t_1)))\end{aligned} \tag{3.16}$$

Rule 6 All outputs must appear as (direct/indirect) inputs on the given destinations.

$$\begin{aligned}&(\forall t_1, o, d).output(t_1, o) \land has-dest(t_1, o, d) \supset \\ &\quad (\exists t_2, i, s).t_2 = d \land input(t_2, i) \land o = i \land has-src(t_2, i, s) \land \\ &\quad ((s = t_1) \lor (ancest-of(s, t_1)))\end{aligned} \tag{3.17}$$

At any point during the viewpoint decomposition phase, a CPM toolset user may run the consistency checks given the representation and rules described above. Errors in the specification are presented to the user which include the type of error and the diagram(s) on which they occur. An example of this output for the supply chain domain is shown in Figure 3.6.

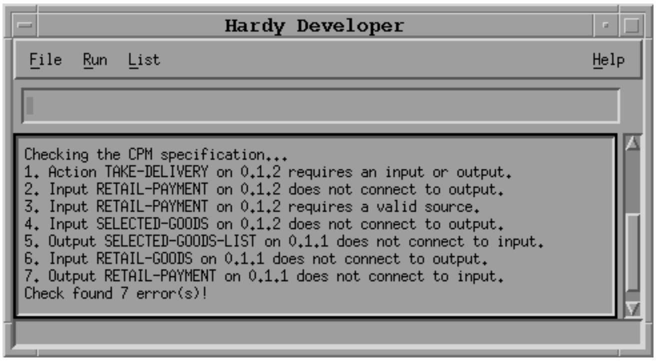

Figure 3.6: CPM Toolset reported errors

3.2.5 Summary

In this part of the framework, we sought to provide a more disciplined approach to producing domain specifications which would encompass requirements capture, analysis and domain requirements engineering. The Common Process Methodology has taken a step toward this goal. We described both the methods and work products along with the toolset used to support the methodology. In Chapter 6 we will return to illustrate the CPM methods for each of our portfolio examples. In [Polyak, 1999] we evaluated this approach against a set of guidelines produced by [Sommerville and Sawyer, 1997] which were considered necessary for any organisation engaging in requirements engineering. This showed that much of what is expected of a requirements engineering approach is supported by CPM. In Chapter 7 we examine some of the advantages and disadvantages of using this methodology with an eye toward improvements which we wish to make in the future.

3.3 Representation

In this section we present the three central components of the shared representation utilised in the Common Process Framework: the Process Ontology (CPO), the Process Language (CPL), and a mechanism for extension. As we shall see, the process ontology is used to provide definitions for the terms and concepts which may be expressed in CPL. Extensions to both the ontology and the language may be made in order to accommodate spe-

cialisation of the framework for particular needs. Following our overview of these components we provide examples of both CPF world and process descriptions.

3.3.1 Process Ontology (CPO)

We begin our presentation of the CPF approach to representation by discussing the shared process ontology. This section is based on our description of CPO which we provided in [Polyak and Tate, 1999]. We will look at the three categories of knowledge in CPO: meta-ontology, object ontology and constraint ontology and show how this approach builds on the <I-N-OVA> *constraint model of activity.*

In Section 2.7.1 we referenced work on ontologies and described some of the possible roles of an ontology. We also described ways that ontologies are being used to support knowledge sharing in Section 2.3.4.2. As part of the intended support provided by CPF involves knowledge sharing (i.e. system integration and communication of a "shared understanding"), we adopted an ontology-centred approach to support this aspect.

In the ontological engineering methodology, Methontology, one of the very first steps prescribed is "specification" [Fernández *et al.*, 1997, Gómez-Pérez *et al.*, 1996]. This step is meant to encourage ontology authors to address questions of purpose and scope straight away. Example issues include: "why is this ontology being built?" and "who are the intended users?". We have begun to outline the purpose and scope of CPO in the introduction (see Section 1.2.3). Using a variety of input sources, we outlined a requirements specification for CPO in order to further delimit the scope [Polyak, 1997b]. These requirements were separated into representational (i.e. what do we want to express?) and functional (i.e. what are the intended uses of the knowledge?). Within each we provided an additional clustering of requirements around concepts (e.g. activities, agents, evaluations, etc.) or uses (e.g. editing, execution, task assignment, etc.). We will present this requirements specification in our analysis in Section 6.1.2.

One of the sources of input for this specification included the set of requirements developed for NIST's manufacturing-based Process Specification Language (PSL)

[Schlenoff *et al.*, 1996]. These requirements were drawn from a range of process management tools and applications. During our initial involvement with the PSL project, we provided an analysis of how well several existing plan/process or activity-based ontologies could address these requirements [Polyak and Tate, 1997]. Tate's <I-N-OVA> constraint model of activity [Tate, 1995, Tate, 1996c] provided the most flexible approach as compared to the others. The <I-N-OVA> model can be seen as a specialisation of an <I-N-CA> shared model [Tate *et al.*, 1999a][4]. Thus, we proceeded with our work being grounded in the <I-N-OVA> approach.

3.3.1.1 <I-N-OVA> Constraint Model of Activity

In order to review a bit from Section 2.3.4.3, we note that the <I-N-OVA> model (Issues, Nodes, Orderings/Variables/Auxiliary) is a means to represent and manipulate plans/processes as a set of constraints. The node constraints in this model set the space of behaviour within which a process may be further constrained. The issues (which could be considered to be implied, to do, or future constraints on behaviour) and remaining constraints (OVA) restrict the processes within that space which are valid. Ordering (O) and variable (V) constraints are distinguished from all other auxiliary (A) constraints since these act as cross-constraints, usually being involved in describing or further restricting the others. By having a clear description of the different components within a process, the model allows for processes to be manipulated and used separately from the environment in which they are generated. For example, we may wish to utilise an artificial intelligence planning system to synthesise a base process given some particular set of objectives and then take that information to a process editor for visualisation or further editing.

3.3.1.2 Plan Ontology

In [Tate, 1996d] an informal plan ontology was developed based on the <I-N-OVA> model. At its heart, this plan ontology envisions a plan as a specialised type of design. As we mentioned back in Section 1.1, while a design for some artifact is considered to be a set of constraints on the relationships between the entities involved in the artifact,

[4] <I-N-CA> stands for Issues, Nodes, Critical and Auxiliary constraints.

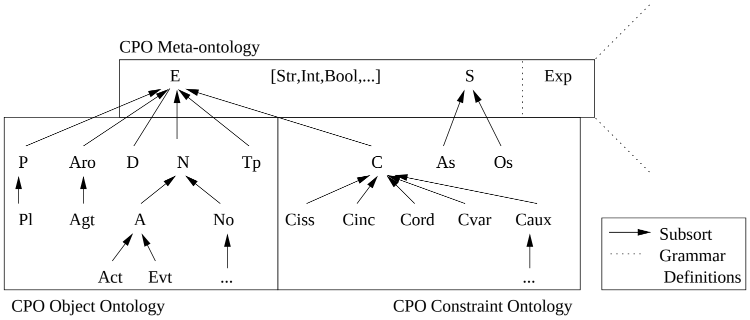

Figure 3.7: 3-CPO: meta, object, and constraint ontology

a plan or process further constraints these relationships to be between agents, their purposes and their behaviour. CPO is very strongly aligned with this ontology and while the above cited work presents the plan ontology primarily in terms of natural language and structured sentences, this thesis work refined and formalised it to be used with a sorted logic in which ontologically-based processes can be expressed. We will now outline this specific set of classes/sorts, functions, and relations[5]. The complete, formal and machine-readable version is expressed using ontolingua [Gruber, 1993] (see Section 2.7) which appears in Appendix A.1.

3.3.1.3 CPO: Core Concepts

The CPO can be separated into three distinct parts, 3-CPO: meta-ontology, object ontology, and the constraint ontology (see Figure 3.7). These elements are considered to be "core" or central to any process description. This "identifying core" approach can be found in the PIF, PSL, and SPAR projects as well (see Section 2.3.4.2). Extensions to any part of 3-CPO can be made in order to customise this set of core elements for specific domains, concepts or applications. These extensions may be packaged into manageable modules to promote shared communication between groups. This is an implementation of the "partial shared view mechanism (PSV)" developed in [Malone and Lee, 1990] which is also used in PIF. We provide examples of extensions to the core in Section 3.3.2.5.

In presenting CPO, we will describe a structured universe of discourse. We assume

[5] A similar approach to communicating constraint information between planners and schedulers has also been investigated in May 1996 by David Joslin at CIRL (University of Oregon).

that this universe can be partitioned into certain sub-universes. Particular relationships exist between pairs of these sub-universes: they may be disjoint, they may have non-empty intersections or one may be completed contained in another. Further, we will state that certain mappings and relations are meaningful only for certain sub-universes. These sub-universes of discourse are given names, called sort symbols.

The set of all sort symbols is partially ordered by a sub-sort order relation, thus expressing the inclusion relationships which hold between sub-universes under consideration. We will refer to a set of sort symbols with the sub-sort order applied to it as a sort hierarchy. These sorts, combined with classical first-order logic (FOL) will give us a many-sorted, or simply, a sorted FOL [Davis, 1990, Cohn, 1985, Walther, 1985] for expressing process knowledge[6]. The implementation of this language in CPF is called the Common Process Language (CPL). The lexicon and grammar for CPL is presented in Appendix B and also appears in [Polyak, 1998b]. We discuss this language definition and the major terms and relationships for CPL in Section 3.3.2.

The following sections overview various CPO function, predicate, and variables symbols which exist within these sub-universes. Sorts will be defined in scripted uppercase (e.g. $\mathcal{P}$ for processes), lower case letters indicate variables of a specific type (e.g. p is a variable of type $\mathcal{P}$) whereas uppercase letters are used for constants (e.g. P is a constant of type $\mathcal{P}$).

3.3.1.4 CPO Meta-ontology

As suggested in Tate's plan ontology approach [Tate, 1996d], the very top of the CPO sort hierarchy is reserved for meta-concepts which help to structure the universe of discourse. Broadly speaking, we can consider the ontology to be composed of a set of entities, a set of data types and a set of relationships between entities. We will define sorts $\mathcal{E}$ and $\mathcal{S}$ for entities and sets along with various elemental and composite data types such as $\mathcal{S}tr$, $\mathcal{I}nt$, and $\mathcal{E}xp$ for strings, integers and expressions, respectively. We will not reify the notion of relation, but we will be referring to various constraint types that have defined entity-relating expressions. In defining various expression types, we will specialise the base expression sort, $\mathcal{E}xp$. Figure 3.7 depicts the relationship between

[6] Sorted logics are very similar to typed programming languages.

the meta-ontology and the constraint and object ontologies.

3.3.1.5 Entities

An entity, $\mathcal{E}$, provides the top-level root for much of the CPO sort hierarchy. In particular, $\mathcal{E}$ may be sub-classed into the sorts $\mathcal{C}, \mathcal{P}, \mathcal{N}, \mathcal{A}ro, \mathcal{T}p, \mathcal{D}$ for constraints, processes, nodes, activity-relatable objects, timepoints and domain levels respectively. This is a slightly different top-level than those found in the PIF, PSL, and SPAR ontologies but is easily related to them.

Earlier, we referred to the sub-sort order relation which provides structure for the sort hierarchy. This relation can be expressed using a simple "isa" predicate. In our implementation, we will use the following notation which expresses, in this case, that a CPO constraint is a CPO entity[7]

$$\mathcal{C} \subset \mathcal{E} = isa(\mathcal{C}, \mathcal{E})$$

3.3.1.6 Sets

One of the basic assumptions of the underlying <I-N-OVA> model is that a plan or process can be represented as a set of constraints on behaviour. At the very least, we require a sort for sets, $\mathcal{S}$, which provides some of the basic set theory relations and functions. For convenience, we will use the following notations

$$\begin{array}{l}
\emptyset = emptyset \\
\{C\} = adjoin(C, emptyset) \\
\{C_1, C_2\} = adjoin(C_1, adjoin(C_2, emptyset)) \\
\{C_1, C_2 | S\} = adjoin(C_1, adjoin(C_2, S)) \\
C \in S = member(C, S) \\
S_1 \cup S_2 = union(S_1, S_2) \\
S_1 \cap S_2 = intersection(S_1, S_2) \\
S_1 \subseteq S_2 = subset(S_1, S_2)
\end{array}$$

3.3.1.7 Strings and Expressions

One of the advantages of the <I-N-OVA> perspective is the identification of various constraint types, i.e. specialisations of $\mathcal{C}$. Each of these constraints express various relationships between CPO objects. As in SPAR, CPO requires extensions which provide "plug-in grammars" that structure constraint expressions. We cited similar examples

[7] Our intended use of "isa" is equivalent to the sub-sort relation defined in [Walther, 1985].

of such flexible approaches in Section 2.7.2. These plug-in expressions will be specialisations of the base $\mathcal{E}xp$ type. We will use the following notation for strings and expressions

$$
\begin{aligned}
&[] = Nil\\
&[Str] = cons(Str, Nil)\\
&[Str_1, Str_2] = cons(Str_1, cons(Str_2, Nil))\\
&[Str_1, Str_2|Exp] = cons(Str_1, cons(Str_2, Exp))
\end{aligned}
$$

Expressions may be composed by concatenations of various strings. These strings may be variable, function, relation, constant or logical symbols. This is analogous to the "PIF-SENTENCE" described in the PIF work [Lee *et al.*, 1998a]. One important predicate that applies to $\mathcal{E}xp$ is a unification evaluation, $unifies(Exp_1, Exp_2)$, which implies that the expressions can be made identical by appropriate substitutions for their variables [Genesereth and Nilsson, 1988].

3.3.1.8 CPO Object Ontology

By "object" ontology, we are mainly indicating those entities which will be involved in and referred to by various constraint expressions which are connected to particular CPO constraint types. For example, the expression of an "ordering" constraint may relate two objects of type $\mathcal{T}p$. These constraint types are described in the constraints ontology which is discussed in Section 3.3.1.16.

3.3.1.9 Processes

Generically speaking, a process, $\mathcal{P}$, provides a **specification** of **behaviour** for some time interval bounded by a pair of begin/end timepoints. By **behaviour**, we mean something that one or more agents perform. The notion of **specification** is simply that of a set, $\mathcal{S}$, of constraints, $\mathcal{C}$. In the constraint ontology, we define this to be an activity specification, $\mathcal{A}s$. We associate these via the relation $activity-spec \subseteq \mathcal{P} \times \mathcal{A}s$. Note that this does not commit to a single As for a given P, although some extensions of this ontology may do so. In addition to this, we observe that it might be the case that $as = \emptyset$ which is interpreted as "do anything". Clearly the opposite extreme may be to specify "do nothing". This may also be accomplished with an $activity-spec(P, as)$ for P which contains a not-include constraint which is discussed in Section 3.3.1.18.

CPO requires certain functions to be defined for all objects of type $\mathcal{P}$. The following two are suggested by the informal description above:

$$start - timepoint : \mathcal{P} \rightarrow \mathcal{T}p$$
$$finish - timepoint : \mathcal{P} \rightarrow \mathcal{T}p$$

Additionally, there are functions defined which support expansion and decompositional relationships between processes and process actions.

$$pattern : \mathcal{P} \rightarrow \mathcal{E}xp$$
$$expands : \mathcal{P} \rightarrow \mathcal{A}$$

The expression returned by the pattern function may be matched with the patterns of various activities and represents its potential to act in a decomposition relationship. An actual decompositional commitment to a particular activity is expressed using the *expands* predicate.

3.3.1.10 Plans

CPO distinguishes a plan, $\mathcal{P}l$, from a process by stating that $\mathcal{P}l \subset \mathcal{P}$ with the additional constraint that a $\mathcal{P}l$ exists for some specified objectives. That is, a plan extends the definition of process to say that it: provides a specification of behaviour **for some objectives** over some time interval bounded by a pair of begin/end timepoints. Objectives, $\mathcal{O}bj$, and objective specifications, $\mathcal{O}s$, are discussed in the constraint ontology section. This is related via a required $objective - spec \subseteq \mathcal{P}l \times \mathcal{O}s$ where $\mathcal{O}s \neq \emptyset$.

3.3.1.11 Nodes

In the overview in Section 3.3.1.1, we referred to the fact that node constraints in the <I-N-OVA> model set the space within which a process may be further constrained. These constraints may either specify that a node is necessarily included or cannot be included at all. The CPO node type, $\mathcal{N}$, referred to is actually an abstract structuring of more specialised CPO concepts: activity, $\mathcal{A}$, or other nodes, $\mathcal{N}o$.

The $\mathcal{N}o$ type is, in turn, another sub-structuring of the domain of discourse. Currently, the only subtypes for $\mathcal{N}o$ are the following "dummy" node types $\mathcal{N}s, \mathcal{N}f, \mathcal{N}b, \mathcal{N}e$ which denote the elements of interval endpoint pairings {Start/$\mathcal{N}s$,Finish/$\mathcal{N}f$} and {Begin/$\mathcal{N}b$,End/$\mathcal{N}e$}. A {$\mathcal{N}s$,$\mathcal{N}f$} pair is, by convention, used for indicating the entire interval for some, possibly decomposed, process or plan whereas {$\mathcal{N}b$,$\mathcal{N}e$} is used

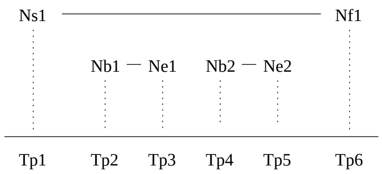

Figure 3.8: Node interval endpoint pairings

to demarcate subprocess intervals. All of these "dummy nodes" listed above represent an instantaneous point and therefore may only be related to a single timepoint. For example, Figure 3.8 indicates two subprocess intervals within an overall process or plan. Future extensions may include additional specialisations of $\mathcal{N}o$ such as: or-split, or-join, and-split, and-join, conditional, iteration, for-each, etc.

3.3.1.12 Activities

For the most part, nodes in a process are used to denote activity. To be more precise, we indicate that $\mathcal{A} \subset \mathcal{N}$ where an $\mathcal{A}$ is meant to represent an activity. As with processes, activities have a temporal extent which is bounded by a begin and end timepoint.

$$begin - timepoint : \mathcal{A} \rightarrow \mathcal{T}p$$
$$end - timepoint : \mathcal{A} \rightarrow \mathcal{T}p$$

Additionally, there are functions defined which support expansion and decompositional relationships for activity. The notion of hierarchical relationships in planning is well established (see Section 2.3.1.3). In fact, it has been pointed out that "planning domains such as errand-running require plans rich with structure. To be useful, abstractions must embody the variable structure of the plans." [Hayes-Roth *et al.*, 1983].

$$pattern : \mathcal{A} \rightarrow \mathcal{E}xp$$
$$expansion : \mathcal{A} \rightarrow \mathcal{P}$$

Notice that this provides a doubly-linked set of decompositional relationships. Given some process, P, we can directly determine which $\mathcal{A}$ it expands (i.e. its abstraction) or conversely, given some activity, A, we can refer to its (possibly null) expansion (i.e. its decomposition), $\mathcal{P}$. Given this information we know

$$(\forall a)(\exists p).expansion(a) = p \supset expands(p) = a \wedge unifies(pattern(p), pattern(a))$$
$$and$$
$$(\forall p)(\exists a).expands(p) = a \supset expansion(a) = p \wedge unifies(pattern(a), pattern(p))$$

In accordance with Tate's plan ontology, we can further specialise $\mathcal{A}$ into $\mathcal{A}ct \subset \mathcal{A}$ and $\mathcal{E}vt \subset \mathcal{A}$. The ontological distinction being made here is between actions, $\mathcal{A}ct$, which are performed by modelled agents and events, $\mathcal{E}vt$, which are performed by an unmodelled agent or agents (this is often referred to as the "environment").

3.3.1.13 Timepoints

A timepoint in CPO, $\mathcal{T}p$, characterises a specific, instantaneous point that lies along a line which is an infinite sequence of time points. Pairs of timepoints for nodes and processes delimit a time interval. In particular, we can use an axiomatisation based on Hayes' catalogue of temporal theories [Hayes, 1996] in order to map timepoints and ordering constraints into Allen's 13 relations between intervals [Allen, 1984a, Allen, 1984b]. During our application of these definitions, we spotted and corrected a couple of errors in this mapping axiomatisation[8].

This axiomatisation is used in the Common Process Assistant (CPA) to map the timepoints and ordering constraints which are passed from the process and domain editing tools, upon a users request, into an interval theory for consistency checking. Allen's table of legal relationships between intervals is then used to detect errors and to provide rationale for why a process specification is incorrect (i.e. CPA explains which legal interval relationships could exist).

In addition to the CPA analysis, the process and domain editors can automatically and efficiently assist users by preventing illegal or unnecessary timepoint constraints between two activities based on the knowledge provided in an $\mathcal{A}s$. As we have pointed out, each $\mathcal{A}$, has 2 timepoints which we will abbreviate as: $\mathcal{T}p^{\mathcal{A}}_{begin}$ and $\mathcal{T}p^{\mathcal{A}}_{end}$. There is one relation that always exists between an activity, A_1, timepoint pair: $before(Tp^{A_1}_{begin}, Tp^{A_1}_{end})$. No other relation can be made between these two points. We will discuss this aspect in more depth in our presentation of CPA in Section 5.4.

[8] The axioms are listed in Appendix C.1 and this is discussed in more detail in Section 5.4 as well as [Polyak, 1998c].

3.3.1.14 Activity-Relatable Objects

In PIF, one of the top-level classes of entity is OBJECT. It is informally defined as "an entity that can be used, created, modified, or used in other relationships to an activity". An identical type is utilised in PSL. During our work on SPAR it was decided that the term "object" was a bit too overloaded and would perhaps confuse the understanding of this class. The more specific Activity-Relatable Object (ARO) term was chosen for SPAR instead. CPO represents this in the sort, $\mathcal{A}ro$.

Some entities of type $\mathcal{A}ro$ are often referred to as "resources". As the PIF definition above suggests, these are the things which are used (e.g. drill, hammer, etc.), modified (e.g. board, metal sheet), etc. This sort also represents those things produced, which might be labelled "products". It is important to note though that these common references or labels tend to be role-defined, which we discussed in [Polyak *et al.*, 1998]. For example, an $\mathcal{A}ro$ for one activity, A_1, may be a "product" for A_1, but it might be a "resource" for A_2.

Subtypes of $\mathcal{A}ro$ are defined for domain-specific applications of CPO (e.g. manufacturing objects might include various drill, saw, lathe types, etc.). As mentioned earlier, these may be packaged into PSV-like extensions to support reuse or to encourage modularity. Domain independent extensions may also be created to provide rich structure between $\mathcal{A}ro$ types (e.g. part-of, requires relation, etc.)

A special agent sub-sort of $\mathcal{A}ro$ has been included in CPO: $\mathcal{A}gt \subset \mathcal{A}ro$. An informal reference for the $\mathcal{A}gt$ sub-sort can be found in the SPAR sentences [Tate, 1998] which refer to it as an "ACTIVITY-RELATABLE-OBJECT which can PERFORM ACTIVITIES and/or HOLD OBJECTIVES". The inclusion of this concept in CPO points to the influence of workflow languages like the WPDL. Specifically, in a process specification, we are interested in knowing who or what will be performing activities and in also linking the purpose of these sets of activity with the agents who held the objectives. As we shall see in the CPO constraint ontology, some aspects of an objective specification are characterised by agent's requirements while others can be considered to be preferences. Thus we have agent relationships

$$performs - activity \subseteq \mathcal{A}gt \times \mathcal{A}$$
$$performs - process \subseteq \mathcal{A}gt \times \mathcal{P}$$
$$has - requirement \subseteq \mathcal{A}gt \times \mathcal{O}s \times \mathcal{P}$$

$has - preference \subseteq \mathcal{A}gt \times \mathcal{O}s \times \mathcal{P}$
$has - requirement \subseteq \mathcal{A}gt \times \mathcal{O}s$
$has - preference \subseteq \mathcal{A}gt \times \mathcal{O}s$
$has - capability \subseteq \mathcal{A}gt \times \mathcal{E}xp$

Note that sets of requirements or preferences may be universal for an agent (e.g. "prefer transportation by boat for any set of activity") or process-specific (e.g. "prefer transportation by airplane for process P_1"). As in subtypes of $\mathcal{A}ro$, the subtypes of $\mathcal{A}gt$ can be specialised for a domain. Domain independent extensions may also be added to provide concepts such as organisational structure (e.g. reports-to, coaches, etc.)

3.3.1.15 Domain Levels

In the Common Process Methodology (CPM) (see Section 3.2 or [Polyak, 1999]), a level-oriented approach to domain modelling is adopted whereby actions, events, effects, and resources are all separated into a series of defined and increasingly detailed levels, $\mathcal{D}$. This helps to avoid the commonly experienced problem of "hierarchical promiscuity" [Wilkins, 1988] or "level promiscuity" which is characterised by the inconsistent usage of various domain elements at varying areas in the overall domain description. This approach is taken directly from our characterisation of the TF Method [Tate *et al.*, 1998b, Tate *et al.*, 1999b].

Domain levels should be assigned meaningful labels which indicate their overall perspective (e.g. "house building task level"). These levels may be structured into a domain lattice and processes assigned to a particular domain level. The following functions and relations partially support these requirements

$label : \mathcal{D} \rightarrow \mathcal{S}tr$
$number : \mathcal{D} \rightarrow \mathcal{I}nt$
$contains \subseteq \mathcal{D} \times \mathcal{P}$

3.3.1.16 CPO Constraint Ontology

In this section, we describe various categories of constraints which may be placed between CPO objects. These constraint types are based on the <I-N-OVA> model and Tate's plan ontology which were introduced above. Primarily we are interested in two types of things: a single constraint, $\mathcal{C}$, and an aggregation of constraints, or a set, $\mathcal{S}$. Also, the expression of a constraint, $\mathcal{E}xp$, for each of the various types is of interest to us, but it will be defined using a highly flexible approach. In order to make this framework generically applicable, we introduce a "plug-in" syntax for expressions as described in the SPAR approach. We provide examples of this below.

Tate describes a constraint as "a relationship which expresses an assertion that can be evaluated with respect to a given plan as something that may hold" [Tate, 1995]. In addition to this, it is pointed out that there is typically a need to recognise which agent added a specific constraint during a design process. At a high-level, we can relate these entities using

$$expression : \mathcal{C} \rightarrow \mathcal{E}xp$$
$$added - by : \mathcal{C} \rightarrow \mathcal{A}gt$$

The design of a process, $\mathcal{P}$, has a relationship with a set of these constraints which denote the process activity. We will refer to this set as an activity specification, $\mathcal{A}s \subset \mathcal{S}$. In addition to this, we further distinguish that a plan, $\mathcal{P}l$ relates an $\mathcal{A}s$ to some set of objectives, $\mathcal{O}s \subset \mathcal{S}$. An objective, $\mathcal{O}bj \subset \mathcal{C}$, may be a requirement (hard constraint) or a preference (soft constraint).

$$member - as \subseteq \mathcal{C} \times \mathcal{A}s$$
$$member - os \subseteq \mathcal{O}bj \times \mathcal{O}s$$
$$soft - hard - info : \mathcal{C} \rightarrow soft, hard$$

The expression of an objective, as with the other constraints, is defined by providing a structuring plug-in grammar. This approach is partially based on the way flexible tasks and goals are expressed in EXPECT [Swartout and Gil, 1996, Swartout and Gil, 1995] and INSPECT [Valente *et al.*, 1996]. We can consider a constraint's expression to be similar to the "Any" type in CORBA IDL [Mowbray and Zahavi, 1995]. Just as objects passed with an "Any" type in CORBA require knowledge of what type the Any reference may be "narrowed" to, constraints in CPF require knowledge of how to narrow an expression to the appropriate constraint expression.

3.3.1.17 Issues

The focus on issues in <I-N-OVA> is a unique approach which is linked to ideas found in workflow perspectives and issue-based collaborative design. Essentially an issue is, "an outstanding aim, objective, preference, task, or flaw which remains to be addressed by the process". Issues refer to "implied constraints" on the actual organisational processes. For example, an issue may refer to an abstract activity which has not been expanded yet, to some condition on an activity which still remains to be achieved, or to some flaw in the overall design of the process.

In CPO, an issue, $\mathcal{C}_{iss} \subset \mathcal{C}$, requires some plug-in syntax which defines the legal grammar for its expression, $\mathcal{E}xp_{iss} \subset \mathcal{E}xp$. For example, we may specify the following structure for an issue (using BNF):

```
<issue-expression> ::= <rtq-sent> | <rt-sent> |
                       <r-sent>
        <rtq-sent> ::= <issue-relconst> <term>
                       [<term>*] <logsent>
         <rt-sent> ::= <issue-relconst> <term>
                       [<term>*]
          <r-sent> ::= <issue-relconst>
  <issue-relconst> ::= "achieve" | "expand"...
         <logsent> ::= {not <sentence>} | etc.
```

Thus, an example of a specific issue constraint, C_{iss1}, which simply states that an activity, A_1 remains to be expanded would indicate $Exp_{iss1} \equiv$ ["expand", "A1"]. This corresponds to the rt-sent defined in the extension.

3.3.1.18 Node Constraints

Node constraints are the backbone of the activity specification constraint set. They provide the space of behaviour on which many of the other constraints seek to further define. Node constraints are so important that a special case has been made for them. Their expression does not require a plug-in syntax, instead there are two built-in functions for declaring that a particular node is either to be included or specifically not to be included

$$include - node : \mathcal{C} \rightarrow \mathcal{N}$$
$$not - include - node : \mathcal{C} \rightarrow \mathcal{N}$$

In fact, there are special cases of both constraints which can be used to refer to an entire class of entities or type. For example, we may wish to specify "do nothing" or "don't do any transportation action". These two concepts use the form $not - include - node : \mathcal{C} \rightarrow \mathcal{S}tr$ where $\mathcal{S}tr$ references a type name (e.g. "cpo-action" or "transport-action").

3.3.1.19 Ordering Constraints

A central aspect of most process specifications is the subset of temporal relationships which define the order in which actions or events will occur. In CPO, this aspect involves those ordering constraints $\mathcal{C}_{ord} \subset \mathcal{C}$. These temporal constraints could be expressed directly between entities of type $\mathcal{N}$ which would be similar to interval relationship approaches (e.g. after, meets, finishes, etc.) but as we will show in Section 5.4, CPO uses a more expressive default ordering approach between timepoints, $\mathcal{T}p$. In particular, part of a default BNF for a $\mathcal{C}_{ord}$ is

```
<ordering-expression> ::= <ordering-relconst>
                          ( <term>, <term> )
   <ordering-relconst> ::= "before" | "equal"
```

3.3.1.20 Variable Constraints

Co-designation and non-co-designation constraints between variables relate activity-relatable-objects in the domain and are quite common in plan and process specifications. These variable constraints, $\mathcal{C}_{var} \subset \mathcal{C}$, limit the range of values which may be assigned to particular variables in CPO expressions. For example, some activity labelled "replace drill bit" may be defined with a pattern "replace-drill-bit ?old ?new". The specification of this activity may include a variable constraint, C_{var1}, which has an expression, Exp_{var1} that specifies that the old bit cannot be the new bit (e.g. Exp_{var1} ≡ ["not_equal_var","(", "?old", ",", "?new", ")"]). Thus part of a default BNF for a $\mathcal{C}_{var}$ is

```
<variable-expression> ::= <variable-relconst>
                          ( <indvar>,  <indvar> )
   <variable-relconst> ::= "equal_var" |
                          "not_equal_var"
```

3.3.1.21 Auxiliary Constraints

Up to this point, we can see that an activity specification, As_1, can be viewed as the union of a set of defined constraint subsets. Specifically we know that

$$\begin{aligned}
&(As_1 \equiv S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5) \Leftrightarrow \\
&\quad (S_1 = \{ciss_1 | ciss_1 \in As1\}) \wedge (S_2 = \{cinc_1 | cinc_1 \in As1\}) \wedge \\
&\quad (S_3 = \{cord_1 | cord_1 \in As1\}) \wedge (S_4 = \{cvar_1 | cvar_1 \in As1\}) \wedge \\
&\quad (S_5 = \{caux_1 | caux_1 \in As1\}))
\end{aligned}$$

The final set that hasn't been addressed yet are the auxiliary constraints, $\mathcal{C}_{aux} \subset \mathcal{C}$, denoted by S_5. This constraint type has a defined sub-sort order structure which is detailed in the CPO ontolingua version. In this section, we will briefly consider some of the common subtypes: $\mathcal{C}_{inp}, \mathcal{C}_{out}, \mathcal{C}_{alw}, \mathcal{C}_{res}$ and $\mathcal{C}_{ann}$.

The first two constraint types, $\mathcal{C}_{inp}$ and $\mathcal{C}_{out}$, relate world state expressions, $\mathcal{E}xp_{ws}$, to particular timepoints. This can be used to express state-based conditions and effects for process activities. The partial grammar outlined below has been used in the CPF for expressions based on the Task Formalism's approach of $< pattern > = < value >$.

```
<world-state-expression> ::= [<ws-type>] <LBRACE>
                             <term>* <RBRACE>
                             [= <term>*]
                             at <term>
                             [<LBRACK> <term>* <RBRACK>]
               <ws-type> ::= "supervised" |
                             "achieve" |
                             "unsupervised" |
                             "only_use_if" |
                             "only_use_for_query"
                <LBRACE> ::= "{"
                <RBRACE> ::= "}"
                <LBRACK> ::= "["
                <RBRACK> ::= "]"
```

So, a particular C_{inp1} which has an $Exp_{ws1} \equiv$ [“supervised”, “{”,“have ?material”,“}”,“at”,“N12”] may depend on a C_{out1} which has an $Exp_{ws2} \equiv$ [“{”,“have bricks”,“}”,“at”,“N10”].

A $\mathcal{C}_{alw}$ constraint differs from those above in that it its assertion is not tied to a particular timepoint, it is defined as always holding in all states. We can modify the $\mathcal{E}xp_{ws}$ grammar above to define a new expression $\mathcal{E}xp_{wsa}$ in which the “$at < term >$” tokens are not required.

The resource constraints, $\mathcal{C}_{res}$, can be used to describe an activity's required allocation of resource objects, producible/consumable resource effects, etc. While it is possible to lump resource constraints into the general notion of input and output constraints it is beneficial to separate them out as many tools are largely geared toward working with this knowledge (e.g. scheduling tools, etc.). The resource expression $Exp_{res1} \equiv$ [“consumes”,“{”,“resource”,“money”,“}”,“=”,“50 pounds ”,“at”,“N10”] may be derived from a grammar which roughly corresponds to

```
<resource-expression> ::= <res-type> <LBRACE>
                          resource <term>*
                          <RBRACE>
                          [= <term>*]
                          at <term>
           <res-type> ::= "consumes" |
                          "produces" etc.
             <LBRACE> ::= "{"
             <RBRACE> ::= "}"
```

Finally, the simplest of these is the annotation constraint, $\mathcal{C}_{ann}$, which can be used to attach unstructured strings to activity specifications. This might be used for attaching additional notes, comments, instructions or possibly to provide links to non-textual or external data related to the process such as CAD and multimedia filenames, web site addresses, or printed policy/standards document references.

3.3.1.22 CPO Summary

Over the last several sections we have introduced some of the components of the core process ontology used in this thesis. We outlined the various sorts and their relationships within the 3-CPO categorisation (see Figure 3.7). What we have seen in past planning literature is that approaches tend to develop highly specialised representations in order to support and indeed optimise for a particular purpose (e.g. plan synthesis, plan evaluation, etc.). Our purpose for CPO is to support a framework which can integrate knowledge from these other specialised representations and provide a rich shared understanding. This is accomplished by providing mappings from application-specific languages into and out of this ontology. We discuss such mappings in Section 5.5. This mapping process or translation requires a well-defined target/source language based on CPO with which world and plan descriptions can be expressed. We will present this language next in Section 3.3.2.

3.3.2 Process Language (CPL)

> *In Section 3.3.1 we introduced a conceptualisation of common process terms and concepts. The component of the CPF which is used to express process knowledge given these terms and concepts is the Common Process Language (CPL). Our implementation of CPL is derived from a sorted, first order language. This language is used to express processes and process domain*

knowledge from a constraint-based view of the world. The language is used by all of the CPF tools for exchanging rich process knowledge. The CPL approach allows for very flexible constraint expressions which are defined via extended grammar specifications. This section presents the CPL and defines both its core lexicon and grammar along with default sub-language extensions for constraint expressions.

The language used to describe domains and processes in CPF is CPL. It includes variables, functions and predicates in addition to the standard logical operators, such as negations, conjunction and quantification. The language is strongly typed, where each type is a finite sort. This section discusses the definition of the CPL language using an extended Backus-Naur form (BNF)[9]. The extended BNF conventions are summarized in the Appendix entry B.1. CPL is built up from a set of basic tokens and certain categories of expressions. Appendix entry B.2 presents these building blocks which are used to define complex CPL expressions. An expression is any string of basic tokens. We define a set of basic expression categories in Appendix entry B.3.

Thus, a <b-con> (i.e., an expression derived from the nonterminal <b-con>) is a string of alphanumeric characters, dashes, and underscores that begins with an upper case letter and in which every dash and underscore is flanked on either side by a letter or digit. A <b-var> is the result of prefixing a <b-con> with a question mark. A <b-func> is just like a <b-con> except that it must begin with either an <oper>, a <punc>, or a lower case letter and it may have a "dot" separator as well (Every <b-pred> is thus a <b-func>). A <doc-string> is the result of quoting any string of tokens; double quotes and the backslash can be used as well as long as they are preceded by a backslash. The $\backslash n$ and $\backslash r$ in the single line comment definition are meant to refer to the newline and carriage return characters.

3.3.2.1 Core Language

The core Common Process Language, L_λ, based on a lexicon λ is the set of all expressions that can be derived from the nonterminal <sentence> in the grammar G_λ. The

[9] This BNF form is borrowed from the style used to describe the enhanced PIF language in "Foundations for Product Realization Process Knowledge Sharing", Knowledge Based Systems, Inc., Final Report, U.S. Dept. of Commerce, Contract No. 50-DKNB-7-90095

members of L_λ will also be called the sentences of L_λ. Given these basic sentence types we will present the lexical elements from S_λ, P_λ, F_λ under conceptual headings. Within the CPF, there are actually two separate categories of knowledge that can be expressed using CPL: process domains or individual processes. We will use the abbreviation CPD for Common Process Domains when we are referring to knowledge in the former category. Some elements of CPL are only appropriate for CPD knowledge while other parts are restricted only to individual process specifications. These restrictions will be pointed out in the description below. Knowledge specifications expressed in CPL are physically identified as files and thus we will use the term, file, to refer to an entire specification or set of sentences. Throughout this section we will provide examples of CPL sentences which will be numbered and listed in bold text. Many of these examples are based on a simple house building domain which is presented in Section 3.3.2.18.

3.3.2.2 Commands

All CPL commands, <command>, start with a percentage sign. These can be used to tell a CPL compiler various things about a domain or process specification. Currently, the only commands defined involve domain definition (%define-domain) which can only be used in a CPD file and domain dependencies (%import-domain) which can be used to express either the link between an individual process file and its domain(s) or when expressed in a CPD, to create a dependency between CPD files.

For example, the first sentence below states that the content of a CPD file can be referred to as the "three_pigs_building" domain. The second sentence would be used in an individual process file to state that a particular process specification applies to the "three_pigs_building" domain. The third sentence might be used to indicate a dependency between some domain and a generic domain which contains a set of building objects.

%define-domain(three_pigs_building) (3.18)
%import-domain(three_pigs_building) (3.19)
%import-domain(general_building_obj) (3.20)

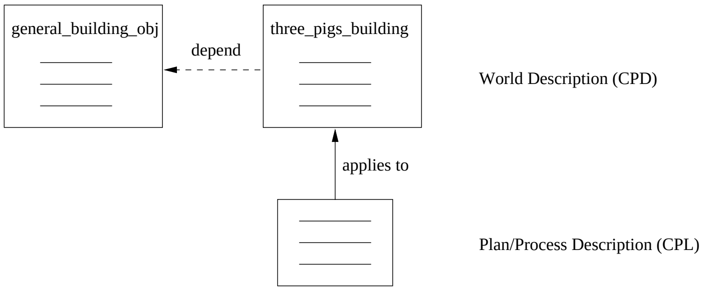

Figure 3.9: Relationships defined via CPL commands

Figure 3.9 illustrates some of these possible relationships. A process description is indicated (CPL file) which is linked to a particular world description (CPD file) which, in turn, is shown to rely on another CPD file (general_building_obj).

3.3.2.3 Sort Definitions

The CPL is strongly typed. The ontology on which CPL is based, CPO (see Section 3.3.1), specifies the sorts of function and predicate parameters as well as the sort of the result in the case of functions. A sort definition sentence, <sortdef>, is used to associate a <con> or a set of <con>s with a particular <sort>.

A (possibly empty) list of named symbols or <con>s can be specified and CPL also provides syntactic sugar for expressing ranges of <con>s more succinctly. For example, sentence 3.21 states that A1, A2 and A3 are of the sort “cpo-action” while 3.22 illustrates a shorthand which could be used as well.

SORT cpo-action={A1,A2,A3} (3.21)
SORT cpo-action={A1-A3} (3.22)

3.3.2.4 Assignments

In CPL, the way to indicate the result of evaluating a function on a domain element (or a specific tuple of domain elements) is to represent it as an assignment, <assignment>. An assignment is given by a function term with parameters, an assignment operator “=”, and the value it should be mapped to. So, for example, we may wish to state that the result of evaluating the function, "activity.begin-timepoint", on the domain element A1 (which is of type "cpo-action", from above) results in the element Tp1, which is defined to be a "cpo-timepoint".

$$\textbf{activity.begin-timepoint(A1)=Tp1} \tag{3.23}$$

3.3.2.5 Extensions

The relationship between the CPO and CPL is completely transparent. Classes in CPO (using the ontolingua definitions) correspond to <sort>s in CPL and the functions and relations from CPO are directly tied to <func>s and <pred>s in CPL. In general, extensions to CPL should be mirrored by and tied to ontological extensions to CPO. There are a couple of very important exceptions to this rule though. The first exception includes those extensions which are only concerned with extended grammar specifications. As we will see, grammar definitions are required to describe the format of various CPL constraint expressions for a particular application of CPL. The definition of these extensions is examined in Section 3.3.2.19.

Another exception permits simple generalisations/specialisations to be declared in a file (without being required to be linked to an external ontology or ontological extension). This can be declared using a simple "entity.isa" assertion in a file. Syntactically this simply relates two <doc-strings>, but it is meant to denote the implied sub-sort order relation. For example, given a three pigs building domain, we may wish to simply add a special activity-relatable-object sort called "pigs-object-material" which will be used to define three material objects which will represent a store of straw, sticks, and bricks in the domain.

$$\textbf{SORT pigs-object-material=\{Mat1,Mat2,Mat3\}} \tag{3.24}$$
$$\textbf{entity.isa("pigs-object-material","cpo-activity-relatable-object")} \tag{3.25}$$
$$\textbf{object.name(Mat1)="straw"} \tag{3.26}$$
$$\textbf{object.name(Mat2)="sticks"} \tag{3.27}$$
$$\textbf{object.name(Mat3)="bricks"} \tag{3.28}$$

3.3.2.6 Process

The first set of CPL constructs (i.e. grouping of sorts, functions, relations) we will present are those related to process. A process sort in CPL is identified by cpo-process.

SORT cpo-process={P1} (3.29)

A process has a start timepoint and a finish timepoint.

process.start-timepoint(P1)=Tp1 (3.30)
process.finish-timepoint(P1)=Tp2 (3.31)

A process can be associated with an activity specification which defines its "space of behaviour". Activity specifications are examined in more detail below.

process.activity-spec(P1)=As1 (3.32)

A process may have a pattern which can be unified with an abstract action pattern to form a decompositional link. For example, the pattern specified in 3.33 unifies with the pattern specified in 3.34 which means that P1 is a potential expansion for Act1. A process may be specified to expand a particular action. For example, sentence 3.35 states that P1 does indeed expand Act1.

process.pattern(P1)="purchase bricks" (3.33)
activity.pattern(Act1)="purchase ?material" (3.34)
process.expands(P1)=Act1 (3.35)

Some processes are considered to be plans which will be identified by a different sort, cpo-plan. Plans carry the additional constraint of being designed for some objectives. This means that a plan will be associated with an objective specification as in sentence 3.37.

$$\textbf{SORT cpo-plan=\{Pl1\}} \tag{3.36}$$
$$\textbf{plan.objective-spec(Pl1)=Os1} \tag{3.37}$$

3.3.2.7 Nodes and Activities

The activity specification linked to a process/plan will typically have constraints which state that certain nodes are to be included (or excluded) from a process/plan. For the most part, these nodes will denote actions or events (which are specialisations of cpo-activity). For example, the action introduced in 3.34 might be declared with

$$\textbf{SORT cpo-action=\{Act1\}} \tag{3.38}$$

In addition to the pattern introduced in 3.34, an activity has a counterpart to the process.expands predicate (e.g. 3.35). This predicate is used to simply state the expansion relationship from the other direction

$$\textbf{node.expansion(Act1)=P1} \tag{3.39}$$

Activities, as with processes or plans, are bounded by two timepoints. To differentiate these from an overall process or plan we use the begin/end pair for activity rather than start/finish.

$$\textbf{activity.begin-timepoint(Act1)=Tp3} \tag{3.40}$$
$$\textbf{activity.end-timepoint(Act1)=Tp4} \tag{3.41}$$

In addition to nodes which represent activity, there are other nodes types which may be declared to help provide structure to the process definition. The most common class of these nodes are the start/finish and begin/end nodes. These are sometimes referred to as "dummy" nodes indicating that they do not denote activity but rather provide structure. The following statements declare particular structuring nodes which may be included in an activity specification

$$\textbf{SORT cpo-start=\{Start1\}} \tag{3.42}$$
$$\textbf{SORT cpo-finish=\{Finish1\}} \tag{3.43}$$
$$\textbf{SORT cpo-begin=\{Beg1\}} \tag{3.44}$$
$$\textbf{SORT cpo-end=\{End1\}} \tag{3.45}$$

All of these node types may be associated with a single timepoint.

$$\textbf{start.timepoint(Start1)=Tp1} \tag{3.46}$$
$$\textbf{start.timepoint(Beg1)=Tp3} \tag{3.47}$$
$$\textbf{finish.timepoint(Finish1)=Tp2} \tag{3.48}$$
$$\textbf{finish.timepoint(End1)=Tp4} \tag{3.49}$$

The "other nodes" type can be extended to provide common process structuring elements such as split/join nodes, and/or, iteration, etc.

3.3.2.8 Activity-Relatable Objects and Agents

Various objects may be involved in or related to process activities. These objects might be employed in various roles. For example, one role might be informally referred to as resource. A resource could be thought of as some object required in order to perform an activity. The objects introduced in 3.24 might be used in this role for a building activity. In general, these objects may be introduced with

$$\textbf{SORT cpo-activity-relatable-object=\{Aro1\}} \tag{3.50}$$

Note that object sort instances can be labelled in order to provide human-readable labels which differentiate their use in the domain (see sentences 3.26 - 3.28). These objects may also be produced, modified, consumed, etc. Constraints in a process' activity specification associate these objects with activities and also indicate the role they are playing.

A special activity-relatable-object is distinguished in CPL. This object represents agents who can perform behaviour, hold purposes, and have capabilities. These objects can be associated with the agent sort

SORT cpo-agent={Agt1} (3.51)

The performs relation for an agent can be expressed as a constraint which is included in an activity specification. This is discussed in the CPL constraint section. The purpose-holding relation on the other hand can be directly assigned between some objective and an agent. This purpose-holding definition may be expressed as a requirement (hard constraint) or preference (soft constraint).

agent.has-requirement(Agt1,Obj1) (3.52)
agent.has-preference(Agt1,Obj2) (3.53)

Capabilities may also be directly associated with agents. The expression of the capabilities is discussed in the constraints section as well (see Section 3.3.2.12).

agent.has-capability(Agt1,Cap1) (3.54)

3.3.2.9 Timepoints

In CPL, the concept of time is approached from a timepoint-based perspective. A cpo-timepoint is an entity that represents a specific, instantaneous point along a timeline which is an infinite sequence of timepoints.

SORT cpo-timepoint={Tp1,Tp2,Tp3,Tp4} (3.55)

Timepoints may be associated with processes or nodes as illustrated in: 3.30, 3.31, 3.40, 3.41. The points may be related with ordering constraints which are discussed in Section 3.3.2.15. Pairs of these timepoints delineate process and activity intervals. In Section 5.4 and [Polyak, 1998c] we discuss the mapping of timepoint-based constraints into interval relationships.

3.3.2.10 Sets

A specification is a fundamental CPL structure which is used to express process design information. Generically speaking, the CPL definition of a specification is simply some set of constraints. When the set of constraints are activity constraints we, call the specification a cpo-activity-specification. When the set of constraints are objective constraints, we call the specification a cpo-objective-specification.

SORT cpo-activity-specification={As1} (3.56)
SORT cpo-objective-specification={Os1} (3.57)

The CPL core supports the most basic set operation which permits a constraint to be specified as a member of the set. For example, 3.58 illustrates an include-node constraint being added to an activity specification (as) and 3.59 illustrates an objective being added to an objective specification (os).

member-as(Ic1,As1) (3.58)
member-os(Obj1,Os1) (3.59)

3.3.2.11 Domain Levels

When CPL is being used to describe a domain (i.e. a CPD file) it is often "good practice" to associate a process with a particular domain modelling level. These level considerations encourage domain modellers to be consistent with their use of domain elements at varying degrees of generality or specificity [Polyak, 1999, Tate *et al.*, 1998b, Tate *et al.*, 1999b].

CPL permits the declaration of domain levels, along with meaningful labels to identify the role of the level, and a numerical value for hierarchically ordering levels.

SORT cpo-domain-level={D1,D2} (3.60)
domain-level.label(D1)="Model house level" (3.61)
domain-level.label(D2)="Primitive building level" (3.62)
domain-level.number(D1)=1 (3.63)
domain-level.number(D2)=2 (3.64)

Processes may then be related to a particular domain modelling level.

domain-level.contains(D1,P1) (3.65)
(3.66)

3.3.2.12 Core Language - Constraints

In this section, we describe various categories of constraints which may be placed between CPO objects. These constraint types are based on the <I-N-OVA> model [Tate, 1995, Tate, 1996c] and Tate's plan ontology [Tate, 1996d]. Tate describes a constraint as "a relationship which expresses an assertion that can be evaluated with respect to a given plan as something that may hold" [Tate, 1996d].

In order to support a range of constraints we present a flexible "plug-in" syntax method for constraint expressions which is similar to the method described in the SPAR approach [Tate, 1998]. We describe some default syntax specifications for most of the constraint types, but these may be modified for a particular use.

constraint.expression(Oc1)="..." (3.67)

There is typically a need to recognise which agent added a specific constraint during a design process. At a high-level, we can relate these entities using

$$\textbf{constraint.added-by(Oc1)=Agt1} \tag{3.68}$$

As we saw in 3.52 and 3.53, constraints may either be labelled as soft or hard depending on the type of purpose held by an agent.

$$\textbf{constraint.soft-hard-information(Oc1)=hard} \tag{3.69}$$

For each constraint type, we will present examples along with a default grammar for expressing the constraint information.

3.3.2.13 Issue Constraints

An issue is an outstanding aim, preference, task, flaw or other issue which remains to be addressed by the process. Issues provide implied constraints on the real world behaviour specified by the process. The default expression of issue constraints will be defined by a verb, zero or more noun phrases and zero or more qualifiers. The initial set of issues may be populated by the objectives set for a plan. The set of issues may expand or shrink throughout the design process. CPL currently considers the expression of objectives and issues to be defined in the same way.

$$\textbf{SORT cpo-objective-constraint=\{Objc1\}} \tag{3.70}$$
$$\textbf{SORT cpo-issue-constraint=\{Is1\}} \tag{3.71}$$
$$\textbf{constraint.expression(Objc1)="expand Act1"} \tag{3.72}$$
$$\textbf{constraint.expression(Is1)="expand Act1"} \tag{3.73}$$

The default definition of an issue constraint expression is

```
<issue-expression> ::= <rtq-sent> | <rt-sent> | <r-sent>
        <rtq-sent> ::= <issue-relconst> <term>+ <boolsent>
         <rt-sent> ::= <issue-relconst> <term>+
          <r-sent> ::= <issue-relconst> <term>*
  <issue-relconst> ::= achieve | expand | add | resolve | evaluate
```

3.3.2.14 Node Constraints

Node constraints form the backbone of a process design. They define the space of behaviour upon which other constraints seek to refine. The primary purpose of these constraints are to specify which actions are to be included in a process. This constraint type is so common that CPL uses a built-in relation as opposed to a plug-in expression type.

$$\textbf{SORT cpo-include-constraint=\{Inc1\}} \tag{3.74}$$
$$\textbf{include-node(Inc1)=Act1} \tag{3.75}$$

Node inclusion is complemented by its counterpart constraint of node exclusion.

$$\textbf{SORT cpo-not-include-constraint=\{Inc1\}} \tag{3.76}$$
$$\textbf{not-include-node(Inc1)=Act1} \tag{3.77}$$

Both the node inclusion and node exclusion relations have unique forms which allow them to refer to an entire sort. This is convenient for saying things like, "no transportation action can be included" or "include any drilling action" or even something as extreme as "do nothing".

$$\textbf{not-include-node(Inc1)=“transport-action”} \tag{3.78}$$
$$\textbf{include-node(Inc1)=“manu-drilling-action”} \tag{3.79}$$
$$\textbf{not-include-node(Inc1)=“cpo-action”} \tag{3.80}$$

3.3.2.15 Ordering Constraints

Ordering constraints may be placed between timepoints in order to define the process temporal structure. The default set of ordering constraint expressions include those which state that one timepoint is before another or that two are equal.

SORT cpo-ordering-constraint={Oc1,Oc2} (3.81)
constraint.expression(Oc1)="before(Tp1,Tp2)" (3.82)
constraint.expression(Oc2)="equal(Tp2,Tp4)" (3.83)

The default definition of an ordering expression is

```
<ordering-expression> ::= <before-sent> | <equal-sent>
        <before-sent> ::= before(<con>,<con>)
         <equal-sent> ::= equal(<con>,<con>)
```

3.3.2.16 Variable Constraints

Co-designation and non-co-designation constraints between variables relate activity relatable objects in the domain and are quite common in plan and process specifications. These variable constraints limit the range of values which may be assigned to particular variables in CPO expressions. For example, some activity labelled "replace drill bit" may be defined with a pattern "replace-drill-bit ?Old ?New". The specification of this activity may include a variable constraint which has an expression that specifies that the old bit cannot be the new bit.

SORT cpo-variable-constraint={Vc1} (3.84)
constraint.expression(Vc1)="not-equal-var(?Old,?New)" (3.85)

The default definition of a variable expression[10] is

```
      <varc-expression> ::= <equal-var-sent> | <not-equal-var-sent> |
                            <equal-vartype-sent> | <not-equal-vartype-sent>
       <equal-var-sent> ::= equal-var(<?var>,{<term>|<doc-string>|<number>})
   <not-equal-var-sent> ::= not-equal-var(<?var>,{<term>|<doc-string>|<number>})
   <equal-vartype-sent> ::= equal-vartype(<?var>,<doc-string>)
<not-equal-vartype-sent> ::= not-equal-vartype(<?var>,<doc-string>)
```

[10] We note that the default ordering and variable constraints from the core are "critical" constraint types which may be present for a range of research purposes (see <I-N-CA> [Tate *et al.*, 1999a]).

Note that this grammar specification is very flexible. While the first parameter in a variable expression is required to be a variable, the second parameter permits several syntactic categories. The second parameter may be a variable, as in the case described above. Other examples include the ability to constrain a variable to be equal or not equal to a certain value (e.g. string or number) or atomic or complex term. The vartype forms allow constraints to be placed on the possible range of values for a variable.

3.3.2.17 Auxiliary Constraints

The auxiliary constraints represent a broad category of constraint types which can be used to detail the design of processes and plans. In this section, we present the current set of core auxiliary constraints which have been defined for CPL.

The first two types provide generic hooks for expressing conditions and effects which may be associated with processes and activities. These are referred to as input and output constraints.

SORT cpo-input-constraint={Ic1} (3.86)
SORT cpo-output-constraint={Oc1} (3.87)

Input and output constraints are used to connect behaviour and state. The expression of both types of constraints is referred to as a world-state-expression. The default approach for representing this knowledge involves stating (pattern)=(value) associations. For example, the process P1 described in 3.33 may have an activity specification which includes some primitive action for buying a supply of bricks. We may wish to state that a condition on the performance of this action is that a supply of money is available beforehand. So, we might include an input-constraint in P1's activity specification which has the following expression

constraint.expression(Ic1)="unsupervised {have money}=true at Tp8" (3.88)

The default definition of world-state-expressions is based on the Task Formalism (TF) [Tate *et al.*, 1994a]. Note that TF allows for typing of these expressions (e.g. unsupervised, supervised, etc.). The following grammar structures these expressions

```
<world-state-expression> ::= [<ws-type>] <lbrace> <term>+ <rbrace>
                             [= {true|false|<term>+}]
                             at <term>
                             [<lbrack> <term>* <rbrack>]
               <ws-type> ::= supervised | achieve | unsupervised |
                             only_use_if | only_use_for_query
                <lbrace> ::= {
                <rbrace> ::= }
                <lbrack> ::= [
                <rbrack> ::= ]
```

While input and output constraints can be used to associate state assertions at particular points in time there are also cases where we may wish to make some assertion apply for an entire domain (i.e. holds for or is automatically included in any domain activity specification). This type of constraint is referred to in CPL as an "always" constraint, as it is in TF. For example, we may assign a wolfproof property to bricks in the three pigs domain.

SORT cpo-always-constraint={Ac1} (3.89)
constraint.expression(Ac1)="{proof_against wolf bricks}=true" (3.90)

The default grammar of an always constraint expression is similar to the world-state-expression defined above with the exception that no <ws-type> or {at <term>} may be used.

The resource constraints can be used to describe an activities' required allocation of resource objects, producible/consumable resource effects, etc. While it is possible to lump resource constraints into the general notion of input and output constraints it is beneficial to separate them out as many tools are geared toward working with this knowledge (e.g. scheduling tools, etc.) For this purpose, CPL has a defined resource-constraint. For example, we may wish to specify that the conclusion of a purchase brick action entails 50 pounds (money) to be consumed.

SORT cpo-resource-constraint={Rc1} (3.91)
constraint.expression(Ac1)="consumes {resource money} = 50 pounds" (3.92)

The default definition of a resource utilisation expression is

```
<resource-expression> ::= <res-type> <lbrace> resource <term>+ <rbrace>
                          [= {true|false|<number>} term*]
                          at <term>
            <ws-type> ::= consumes | produces | uses
             <lbrace> ::= {
             <rbrace> ::= }
```

Finally, an auxiliary constraint may be utilised for attaching annotations or documentation to the process artifact. This could also be used to provide links to non-textual or external data related to a process such as CAD and multimedia filenames, web site addresses, or printed policy/standards document references.

$$\textbf{SORT cpo-annotation-constraint=\{Anc1\}} \tag{3.93}$$

3.3.2.18 Example: Three Pigs Building

The example presented in this section is based on a demonstration building scenario and illustrates the use of the elements we have discussed to this point. This building domain is similar to the Task Formalism three pigs domain which was created for demonstrations of the O-Plan planner. We will explore this domain in Section 6.2.1. The only task in the domain is concerned with building a house for a pig. As in the TF domain, the main building materials involve straw, sticks, and bricks which each cost different amounts of money. Brick material provides security. There are also costs for performing the activities and for other house materials such as windows and doors.

In order to provide a detailed, yet concise example of a CPL process specification which utilises CPO terms and concepts, we will restrict the content to a rather simplified part of the process domain. The example "Purchase Brick Process" is part of the larger 3 pigs building domain and represents a particular transaction activity whereby money is consumed to acquire some supply of brick building material. As we can see in Figure 3.10, it is bounded by a begin/end node pairing and contains only one action, "purchase bricks". We present the example CPL expression of this process in Appendix entry B.5.

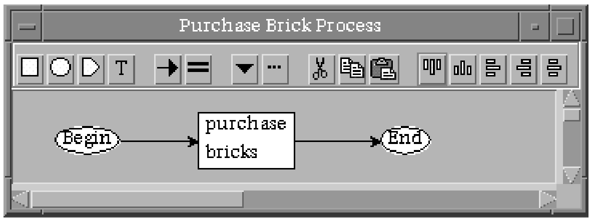

Figure 3.10: Simple CPL process example

3.3.2.19 Extensions: Tool-Based

CPO provides a core set of concepts which may be extended to capture specialised process-related knowledge. One class of extensions can be considered to be tool-specific or tool-based. Tool-specific extensions are used to express new or specialised sorts or relations which address aspects linked to a particular tool's ontology. Two examples are provided here for extensions related to O-Plan's TF and the process/domain editors in CPF.

O-Plan's Task Formalism language [Tate *et al.*, 1994a] encompasses a large set of terms and concepts for expressing plan/process domain knowledge. For this particular TF extension example though, we are simply interested in providing additional support for capturing TF resource-related information. One facet of this information is "resource units". Resource unit statements in TF are used to define unit types for resources such as person/people, gallons, kilograms, etc. These units have their own properties in TF (e.g. type, which could have the values: count, size, weight, or set).

In the TF extension, we define a new sort called resource unit. Two new functions are designated for this sort to express both its label (e.g. pounds) and its type (e.g. count).

$$\textbf{SORT tf-resource-unit=\{Ru1\}} \tag{3.94}$$
$$\textbf{ru-label(Ru1)=``pounds''} \tag{3.95}$$
$$\textbf{ru-type(Ru1)=``count''} \tag{3.96}$$

The ru-label can be any <doc-string> but the ru-type expression above is syntactically constrained to {count|size|weight|set}. In addition to this, we need to add functions and a relation to the activity-relatable object sort. In particular, we need to be able to express whether an ARO is going to play the role of a TF resource and if so, what its TF resource type is

is-resource(Aro1) (3.97)
resource-type(Aro1)="consumable_strictly" (3.98)
unit(Aro1,Ru1) (3.99)

The resource-type expression above is syntactically constrained to the following forms which are defined in the TF manual.

{consumable_strictly | consumable_producible_by_agent |
consumable_producible_outwith_agent |
consumable_producible_by_and_outwith_agent |
reusable_non_sharable | reusable_sharable_independently |
sharable_synchronously}

Some tool-specific extensions are related to presentation information or internal state information (e.g. node positions, nodes selected, etc.) associated with processes. In both the Common Domain Editor (CDE) and the Common Process Editor (CPE) in CPF, process presentation information is attached to various parts of the domain specification. The CPF tools extension defines additional support for this such as

cpf-proc-xpos(P1)=10
cpf-proc-ypos(P1)=10
cpf-proc-width(P1)=400
cpf-proc-height(P1)=400
cpf-proc-label(P1)="Purchase Brick Process"
cpf-node-xpos(Act1)=10
cpf-node-ypos(Act1)=10
cpf-node-type(Act1)="Act"
cpf-node-status(Act1)=0
cpf-node-label="purchase bricks"
cpf-ann-xpos(Anc1)=20
cpf-ann-ypos(Anc1)=20
cpf-proc-top-level(P1)
cpf-node-selected(Act2)

The cpf-node-type may be {Act|Event|Special} which indicates its presentation style. The node status can be used to attach an executability status to nodes. The cpf-node-status can be {0=No information, 1=Complete, 2=Executing, 3=Possible, 4=Impossible}.

3.3.2.20 Extensions: Rationale

While the extensions discussed in the previous section were labelled tool-specific, we can also have extensions which are tool-independent, or more appropriately, concept-specific. Concept-specific extensions provide terms and definitions which are centred around a general set of closely associated entities and relations. One example of such an extension is the rationale extension we have developed for CPO.

In our work with plan rationale [Polyak and Tate, 1998], we explored the epistemological nature of this category of knowledge and described it from the perspectives of dependencies, causal relationships and decisions. While there has been much work done on both plan/process causality and dependencies, there has been correspondingly less research into plan decision rationale (see Section 2.5).

We proposed a "design space analysis (DSA)" approach to plan decision rationale [Polyak, 1998a] which was based on research from the design rationale (DR) field. If we envision the <I-N-OVA> approach, which CPO has adopted, as describing a "space of behaviour" we can also consider a "space of decisions" which is navigated in creating this behavioural specification. It is possible then to augment a process description with the rationale that went into designing this artifact. We will discuss this approach in more detail in Chapter 4.

Both CPE and CDE support this DSA approach (i.e. provide graphical editing of a DSA) and rely on this CPF rationale extension to define the DSA terms and concepts which are expressed in CPL. In this extension, we refer to an entity called a decision rationale which represents the overall "decision space" for a process design. In the CPO core, an activity specification groups the constraints which form the "space of behaviour". Analogously, a rationale specification groups the constraints which form the "space of decisions". So, the CPF rationale extension includes

SORT dsa-decision-rationale={Dr1} (3.100)
SORT dsa-rationale-specification={Rs1} (3.101)
dr.rationale-spec(Dr1)=Rs1 (3.102)
process.decision-rationale(P1)=Dr1 (3.103)

While a plan is described in Tate's plan ontology as "a set of constraints on the relationships between agents, their purposes and their behaviour" a decision rationale can be viewed as "a set of constraints on the relationships between questions (or design issues), options (or answers to these questions), and evaluative criteria". The CPF rationale extension includes the sorts for questions, options and criteria.

Questions pose key issues for structuring the space of alternatives (options). The role of questions is to define local contexts in a design space which help to ensure that certain options are compared with each other. Criteria represent the desirable properties of the process and requirements that it must satisfy. They form the basis against which to evaluate the options. These elements can be included into a rationale specification and interrelated via a set of defined constraints which represent relationships.

SORT dsa-question={Q1,Q2} (3.104)
SORT dsa-option={Opt1,Opt2} (3.105)
SORT dsa-criteria={Crt1} (3.106)
dsa-has-option(Q1,Opt1) (3.107)
dsa-has-option(Q1,Opt2) (3.108)
dsa-selected(Opt1) (3.109)
dsa-supports(Crt1,Opt1) (3.110)
dsa-detracts(Crt1,Opt2) (3.111)
dsa-sub-question(Opt1,Q2) (3.112)
(3.113)

3.3.2.21 CPL Summary

In this section we have described the Common Process Language and provided a set of sentence examples which illustrate the expression of various process and process-related concepts. The CPL provides a concrete, well-defined medium for utilising the core process ontology which was directly derived from the work on the <I-N-OVA> constraint model of activity. As we mentioned, the sorted FOL approach which underpins CPL is based on Dave Joslin's work on establishing a language for exchanging knowledge between planning and scheduling systems. In addition, this work has benefited from experience gained during our collaboration on the default process specification language for the NIST PSL project.

The CPL is unique in that it tailors for flexibility (see Section 2.7.2). We believe that an approach to process specification requires a representation that is extensible and customisable. Such a rich plan/process representations should be built with an eye toward translation or knowledge exchange. There needs to be support for building the knowledge required in disciplined, applied ways. There also needs to be an integrated notion of the "planning process" built in. That is, to be effective, the planning system must be capable of being viewed as one part in a larger agency of individuals working cooperatively to solve the problem. We believe that the CPL is such a vehicle for meeting these needs.

3.4 Conclusion

In this chapter we have explored our research methodology and the design of the Common Process Framework. We described our implementation of the requirements analysis phase which picked up from earlier work on incorporating a set of methods from requirements engineering. We explored both the Common Process Ontology and the language which gives us the machinery for expressing a space of behaviour. In Chapter 4 we turn our attention to the space of decisions and our implementation of a design space analysis to be used with an AI planning approach. We will unite these spaces when we look at the presentation of this knowledge in the CPF toolset in Chapter 5.

Chapter 4

Process Design Space

In Section 2.5 of the literature review we discussed research on design rationale (DR). As we have adopted a design-based perspective on the expression of process knowledge we are interested in establishing a method in which DR can be used to enrich process representations with rationale knowledge. Our motivation was to join both knowledge of the space of decisions with the space of behaviour. In Section 3.3.2.20 we showed how such knowledge can be viewed as a conceptual extension to the core process ontology. In this chapter we describe the aspects of this conceptual extension in our adaptation of the design space analysis DR approach.

The traditional solution produced by an artificial intelligence planning system is a set of actions and ordering constraints. This result is the minimum output required to enact a plan but it represents only one component in a "complete" planning solution. The definition of a complete solution is drawn from work generated by the KADS-II project [Breuker and van de Velde, 1994] which is discussed in more detail in Section 4.1. An adaptation of this definition considers a complete planning solution to be one that contains

- a resultant plan
- a context in which the plan applies
- an argument structure that justifies the plan

The argument structure for a plan generated by an AI planning system is typically omitted from the solution. This omission limits the usefulness of the result and constrains the way a plan can be manipulated and reasoned about throughout the lifecycle of a plan. This argument structure represents the main component that is addressed in this chapter. While complete solutions are not always necessary, increasing demands are being placed on solution representations for real-world planning situations. Richer knowledge about the planning process is sometimes needed to address organisational and environmental issues in these settings. The uses of a "batch solution" which is created by a sole planning agent, as well as an "incremental solution" which supports multi-perspective, mixed-initiative plan argumentation with multiple planning agents shall be considered.

In formulating an approach toward representing and communicating a complete solution, Tate's perspective of a plan as a specialised type of design [Tate, 1996d] is utilised. Researchers in the design community have produced a number of methods and notations pertaining to the explicit representation of design rationale (DR) (See Section 2.5). Since design rationale provides the argument structure for a design artifact, we felt that it would be fitting to apply these methods to planning as well. A previous paper pointed out the similarities between one such DR notation, QOC, and planning decision rationale [Polyak and Tate, 1998]. The approach behind this notation is called "design space analysis" which focuses on the output of a design as a design space rather than a single artifact [MacLean *et al.*, 1991]. We adapted this approach for planning in a system, Nonlin+DR [Polyak, 1998a], using the University of Maryland's release of UM Nonlin [Ghosh *et al.*, 1992, Tate, 1977]. Nonlin+DR supplements a plan solution with an externalisation of the planning decision rationale. The output produced by this prototype system for a simple domain problem is reviewed in Sections 4.3.1.1 to 4.3.1.4.

Our first step is to present our definition of a complete solution as it is applied to planning. Next, the perspective of planning as a specialised type of design activity is considered. The application of the design space approach is reviewed and the prototype implementation, Nonlin+DR is presented and discussed. The ways that Nonlin+DR could be used to assist in the overall planning process and possible directions which lie ahead are examined. As we shall see in Chapter 5, this approach has been integrated into the CPF domain and process editors.

4.1 What is a Complete Solution?

This definition is partially based on Newell and Simon's observation that the concept of a solution typically means different things in various situations [Newell and Simon, 1972]. In their paper, a distinction is made between solution-objects, solution-paths, and solution-actions. A solution object is the direct result that one is typically interested in achieving. For example, in planning this would be actions and orderings and in diagnosis it would be a set of faulty components. Solution paths on the other hand consider the line of reasoning itself to be the focus. This can be seen as the result of a mathematical proof. The emphasis is not on arriving at the outcome hypothesised, but rather the way it was argued. Solution actions are plans or instructions that lead to required solutions and can be considered to be special case "solution objects". Based on this distinction and other sources, Breuker defines a complete solution as one that contains a case model, conclusion, and argument structure [Breuker, 1994].

- Case Model - the understanding or conceptualisation of the problem.
- Conclusion - the answer to the question posed by the problem definition.
- Argument Structure - the reasons why the conclusion is supported.

In terms of planning, the case model is typically embodied by the domain knowledge and structure of the task assignment for a planning problem. The conclusion can be generally equated to the resultant plan. In most cases the argument structure is omitted or "compiled out" of the solution. As we have stated, while complete solutions may not be necessary in artificial settings, they are often required for real-world planning systems. We point out the need for this type of knowledge in two different planning approaches. On one hand, we consider a planning agent that plans in isolation (i.e. stand-alone), and on the other we examine the requirements that are placed on a planning agent involved in mixed-initiative planning (see Section 2.3.3).

4.1.1 Planning Decision Rationale

In our review of rationale in planning, we described a dimension of planning decision rationale [Polyak and Tate, 1998]. Decision rationale is the recording of the reasons why a specific decision was made in a particular way. Recording the rationale of these decisions adds value to the planning process in the following ways:

- facilitation of communication and reasoning
- promoting a shared understanding of beliefs and intentions
- maintaining a consistent approach
- connecting agents to their responsibility in the plan process
- helping to steer the decision-making process

Planning systems that are situated in an organisation must work in cooperation with a variety of agents. This may mean that humans and machines collaborate in the development and management of plans while sharing a common initiative. This has been termed "mixed-initiative planning". With a large number of people and systems working together to produce a solution, there is often a need to communicate intentions, beliefs, and justifications. When a decision is to be made, machine or human, the ramifications need to be considered within a "shared understanding".

In Section 2.3.3, we cited a human planner communication study [Gross *et al.*, 1993]. As you may recall, in no case did the planners simply convey the plan as a set of actions. The agents identified goals and sub-goals, identified important actions, stated relevant facts that would help in the development of the plan, identified problems with what the other agent proposed, requested clarification, confirmed each others suggestions. We feel that this suggests that a richer model of plans is necessary to convey key pieces of knowledge needed to make planning decisions when human beings are involved. An "incremental solution" that contains this rationale could be open to argumentation, inspection, and justified modification throughout the planning process.

Rationale is also important in understanding and using a single agent planning system. This solution is considered to be "batch" in that the decision rationale is recorded in isolation and then is made available at the conclusion of plan construction along with the resultant plan. The types of decisions made by a single agent planning system are limited by the specific refinement methods that it can use. Understanding which refinement method was applied at various stages sheds light on the result of the planning process and opens new avenues of reasoning about the artifact.

Much of what has been said here about planning also applies to design. Designers cooperate by sharing rationale and often need to look behind the artifact to understand the deeper meanings behind the constructs. The research that has addressed this need in the design community is called Design Rationale (DR) (see Section 2.5).

4.2 Design Space Analysis

One DR approach called the design space analysis (DSA) method which underlies the QOC semi-formal DR notation [MacLean *et al.*, 1991] was selected for the implementation of Nonlin+DR. One of the main reasons for this choice was a similarity that can be seen between this approach and perspectives on how plans are built. We have defined QOC in the following way. Assume the existence of a finite set I of questions $\{Q_1, Q_2, ..., Q_n\}$ which reflect choices in the design/plan. Assume also a finite set J of options $\{O_1, O_2, ..., O_m\}$ and a finite set K of criteria $\{C_1, C_2, ..., C_l\}$. Options provide alternatives $alt(O_j, Q_i)$ to questions posed during planning/design. Evaluative criteria may be be attached to options via an assessment relationship $a^+(C_k, O_j)$ or $a^-(C_k, O_j)$ which reflects whether the criteria either supports or detracts from the option. Additionally, a relationship may exist between options and questions in which the question is a sub-issue of an option $s(Q_i, O_j)$. Thus, a DSA is composed of $(I, J, K, \kappa, \lambda, \sigma)$ where κ is the set of alternative relations, λ is the set of assessments, and σ is a set of sub-issue relations. Figure 4.1 shows the general structure of a QOC diagram. QOC can be presented as a node-arc graph where the nodes are **Q**uestions, **O**ptions, and **C**riteria. The relations between these entities is expressed as arcs connecting the nodes.

Another reason for using QOC in Nonlin+DR is the flexibility and simplicity of the notation. Our emphasis was on a representation that succinctly expresses the important

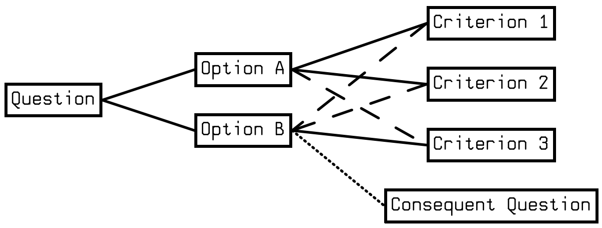

Figure 4.1: QOC, semi-formal notation to represent a design space. Dashed arcs between options and criteria denote negative influence whereas solid arcs indicate positive influence (i.e. arguing for or against an option).

relationships and does not require cumbersome inspection of the details or symbology. An empirical study of designers using QOC showed that designers required low amounts of training to productively use QOC [Buckingham Shum, 1991b] for design tasks. The DSA perspective, along with its simple, straight-forward presentation supports intuitive browsing to answer questions like: What are the other alternatives for this plan? How does criteria from one alternative affect another? What are the tradeoffs among them? etc.

DSA explains design rationale as defining how a given artifact is located in the space of possible design alternatives. Sets of these structures collectively define a "design space" of possible design realizations. This process of "design space" elaboration is similar to the work performed in planning. Tate stresses the importance of issues in his <I-N-OVA> framework [Tate, 1995, Tate, 1996c] which could be mapped to the use of questions in QOC. At a high level, a planning session could be defined by the issues (questions) considered (achieving a goal, assigning a resource, ordering nodes, etc.), the alternatives (options) posed (use operator A or B or C) and the justification (criteria) for those choices (using operator B requires less resource commitment). As it was pointed out before, this externalisation of the planning process is not something that is typically produced in most planners today[1].

As these uses illustrate, representations are now required which weave together expertise on a variety of topics, techniques, and standards involved in complex domains. In each of these applications of AI-based plan representations we can see a set of rich

[1] Exceptions to this include O-Plan [Currie and Tate, 1991] which incorporates this as a design feature and research on explicit meta-plan driven systems.

plan/process elements at the core (e.g. the CPO concepts and terms from Section 3.3.1). This core may not only entail knowledge about the possible elaborations of behaviour that are valid for the plan specification (i.e. the artifact) but also knowledge about the planning, modelling, or (re)design process itself. For example, we may wish to capture and relate knowledge from both the space of decisions as well as the space of behaviour as we indicated in Figure 1.3 on page 11.

In that diagram, decisions are represented by ellipses and boxes represent alternatives. Alternatives considered and selected in the decision space define new boundaries of possible actions in the behaviour space. These spaces are connected in part by the issues that drive this process.

4.3 Recording Planning Decisions

In this section, our initial work on a prototype system is described which was designed to automatically record planning DSA rationale. A plan is contextualised as a specific elaboration in the possible space of planning decisions. This DSA method can be used to support activities in both mixed-initiative and classical AI planning (stand alone) settings. Currently the system only addresses a stand alone approach, but its mixed-initiative potential is examined in Section 4.4.

4.3.1 Nonlin+DR

A design space analysis approach was implemented using the publicly available University of Maryland release of UM Nonlin [Ghosh *et al.*, 1992]. UM Nonlin is a Common Lisp implementation of some aspects of Nonlin, a hierarchical, partially-ordered, domain-independent planning system that was originally developed by Tate [Tate, 1977].

This version, entitled Nonlin+DR, is capable of producing semi-formal rationale output in graph description language (GDL) [Sander, 1994]. GDL output can be visualised using the publicly available tool, XVCG (X-windows Visualisation of Compiler Graphs) [Sander, 1995]. XVCG provides automatic formatting of the design space graphs expressed in GDL and effective management of high-level browsing with built-in interactive scaling. A visual interface for this core planning system was created

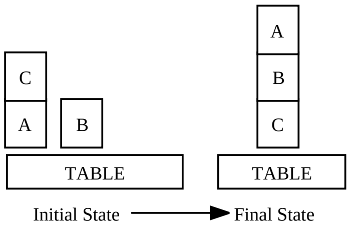

Figure 4.2: Task Formulation for the Sussman anomaly problem

using TCL/Tk [Ousterhout, 1994]. This interface integrates simple task selection, option configuration, and viewing of the plan and associated rationale. An example view of this tool is provided as Figure 4.3.

Currently, Nonlin+DR can be used in a classical AI "batch solution" mode. Once the planning process is complete it exports the recorded decision rationale to be presented by the XVCG tool. The rationale is composed of a set of local decision space graphs. The global decision space can be conceptualised as an aggregation of local decision spaces. Each local decision space maps to the processing of a single issue or agenda item. A review of a simple "sussman anomaly" planning problem [Sussman, 1974] will help to explain this approach.

A standard blocks-world domain is used for this example. In this domain there are two operators corresponding to higher level "operator" schemas: makeon and makeclear. One primitive action schema, puton, is used to define low level activity[2]. The task that is sent to the planner is shown in Figure 4.2.

This is the classic sussman anomaly which is a conjunction of two interacting goals. The problem is typically used in AI planning to show that the simple "linear" approach to solving the two goals in any order will fail. The first local design space generated by Nonlin+DR is represented in Figure 4.4.

4.3.1.1 Select Issue

The first decision that Nonlin+DR was faced with was which goal to work on. The alternatives considered are connected to the right of the decision. At this point, the planner was able to either select (on a b) or (on b c). Nonlin+DR does not have a

[2] See the UM Nonlin manual [Ghosh *et al.*, 1992] for more detail on these operators

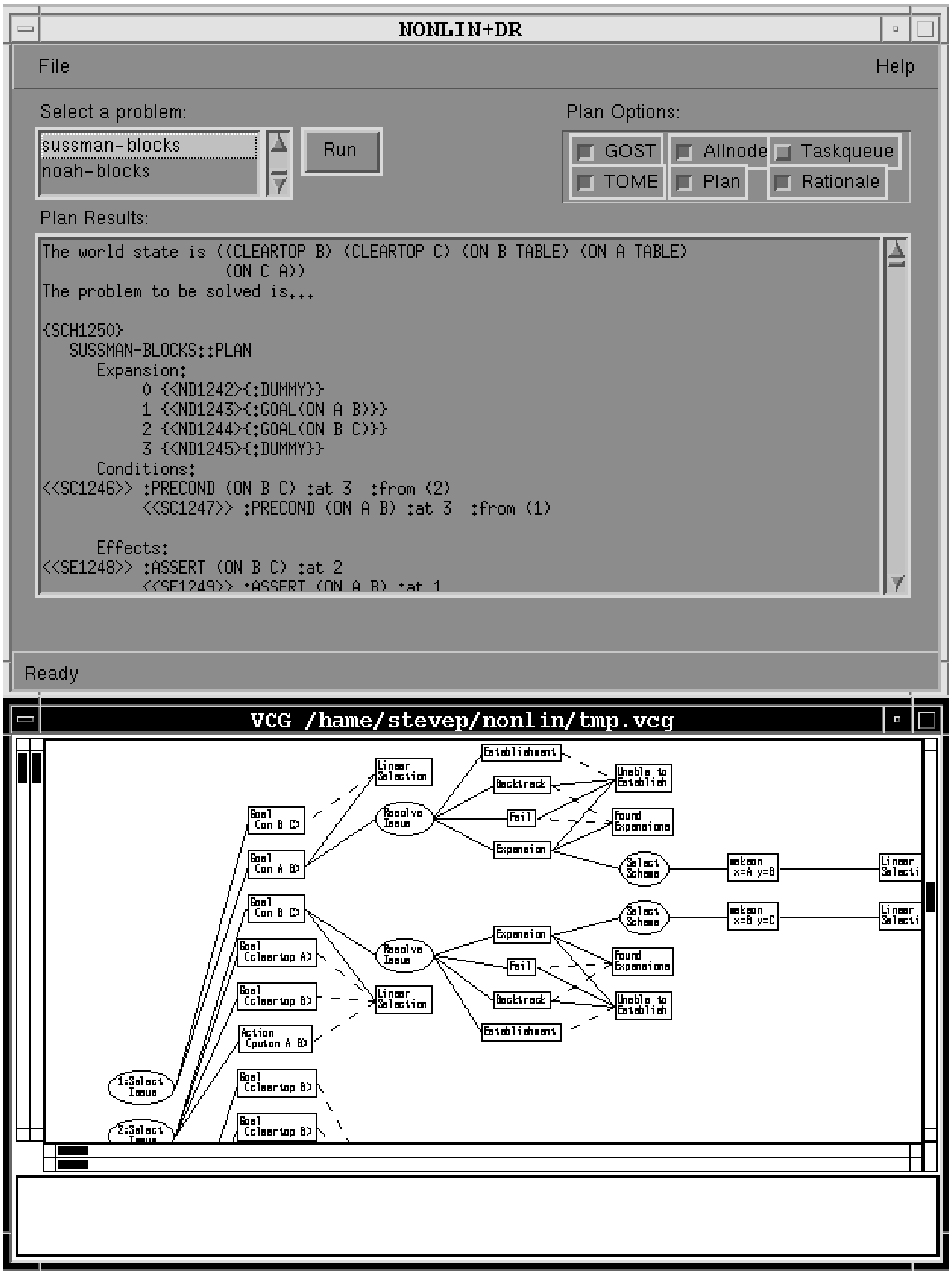

Figure 4.3: Nonlin+DR interface for recording and presenting plan decisions

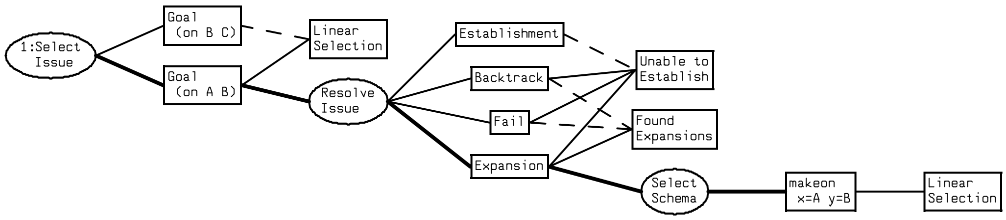

Figure 4.4: Nonlin+DR local design space for processing a single agenda item

very sophisticated mechanism for agenda selection as it only relies on one very basic criteria, linear selection. The algorithm is hard-wired to always process these items in a FIFO manner and is unable to treat this decision opportunistically. This is modelled as a single decision criteria that has an influence on each item in the agenda. Solid criteria links represent positive influence and dashed links represent negative influence (i.e. arguing for or against an alternative). In this case, linear selection criteria will always argue for the first in line and against all others. A bold link from a decision to an alternative indicates the selected course of action. In this case (on a b) is selected. A bold link that carries on from a selected alternative indicates the deliberation of a subsequent decision.

4.3.1.2 Resolve Issue

In this local design space, Nonlin+DR next considered how to resolve the issue. At a high level, the alternatives for resolving a goal are establishment or expansion [Tate, 1977]. It is also possible that the planner may decide to backtrack or fail at this point as well. The planner considered the argument for establishment and realized that there is no support for this. Nonlin+DR records this criteria as arguing against establishment and favouring expansion, backtracking, or failing. When considering the expansion option the planner noted that there was at least one expansion that corresponded to the goal. This favoured expansion over backtracking or failure. The selection to expand then lead the planner to another, rather simple, decision of how to expand.

4.3.1.3 Select Schema

Since there was only one possibility the planner chose it as the way to update the plan in progress. Even if there was more than one way to perform this expansion the decision would still have been very straight-forward because the schema selection only considers linear selection criteria again. An update may add items to the agenda as it does in this case. The planner then moved on to select the next agenda entry which is then described in the next local design space.

Note that the alternatives for an expansion also contain the variable bindings selected for the schema. Expansion alternatives may be due to different schemas that have the same ":todo" pattern but they may also be different instances of the same schema with different bindings. For example, consider the way that the planner addressed the goal "(cleartop A)" in Figure 4.5. For the "select schema" decision, the planner had the choice of either placing C on B or placing C on the table. The table was chosen because this variable binding set was ordered before the other alternative. This was rather fortunate because if the variable binding for B was selected instead it would have led to an inefficient plan where C was unstacked onto B and then subsequently unstacked onto the table.

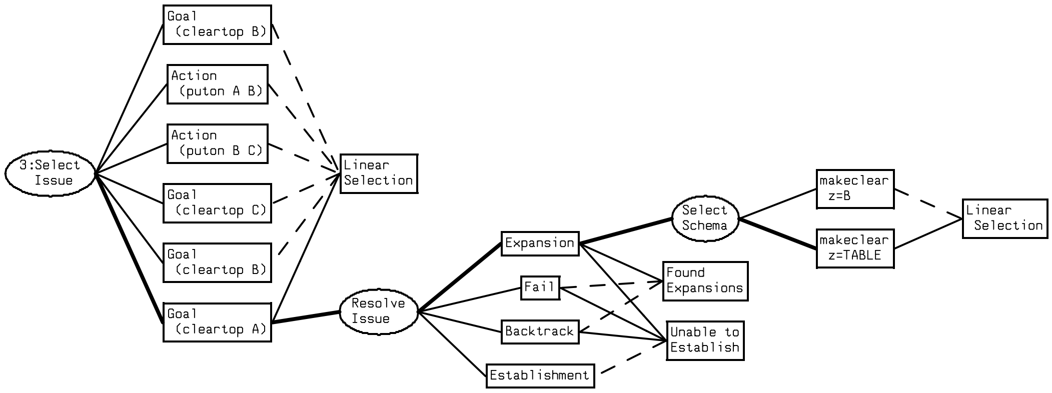

Figure 4.5: Design space resulting from different variable binding choices

4.3.1.4 Resolve Conflict

In Figure 4.6 the schema "puton" was selected to address the "(puton A B)" issue. The planner detected a conflict between an effect from this proposed action and a condition in another part of the plan. Specifically, this action would negate "(cleartop B)" needed to place B on C. In order to utilise this schema, the planner had to make a subsequent decision on how to resolve this conflict. Thus we see that the design space is further defined by alternatives for conflict resolution. These alternatives are either: link "(puton B C)" before "(puton A B)" or link "(puton A B)" before "(cleartop B)". In this case, the planner chooses to link the stacking of B C before the stacking of A B. Again this was a straight-forward linear selection from a list of possible ways to address this problem.

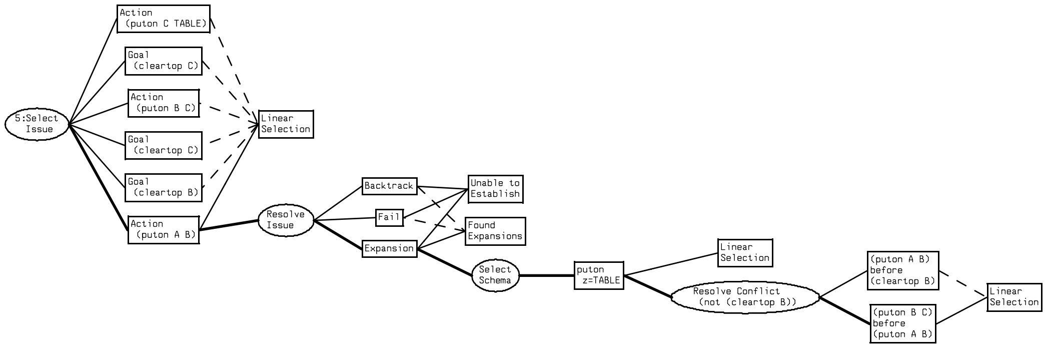

Figure 4.6: Conflict resolution in the design space

The agenda shrinks and grows until all of the items have been processed. Each local design space shows how an agenda item was selected and processed and the high-level criteria that was used to make the selections. Thus, the global design space is an aggregation of the local design spaces explored for each agenda item and represents the overall decision rationale of the plan.

4.4 Nonlin+DR Discussion

This example used here is rather simplistic in two respects. Firstly, this blocks world domain is particularly sparse and does not offer much in the way of "interesting" alternatives. Secondly, the underlying UM Nonlin planner considers only very basic criteria for option selections (e.g. agenda selection, schema instance selection, etc.). The focus of this example though was to clearly explain how DSA could be applied in a basic classical planning session before moving on to more challenging domains and planners. Work on this example produced a list of items to consider and has shown potential issues which need to be addressed when scaling up this approach for more difficult domains and sophisticated planning situations.

Items for future work included an enumeration of a wider set of decisions that are made by planning agents (humans or machines). Some of these decisions naturally come out of a move toward richer situations (e.g. selecting a resource, associating a task executor, etc.). We also believe that DSA may be used to show how various planning systems utilise different approaches[3] and criteria (e.g. linear selection, random selection, smart selection) for the same problems.

A determining factor for this progression will be its application to mixed-initiative planning. The design space approach is seen as a unique way of placing the plan in its broader context. This context could help to focus mixed-initiative discussion on the relevant alternatives and criteria for a specific part of the plan. It may also indicate criteria/alternative interaction that was unforeseen or alternatives that may have been left undiscovered. In order to achieve this level of interaction though it will be necessary to open up the planning interface to allow a user or group of users to control and inspect the planning choices during plan generation. This is similar to what has been done for Prodigy/Analogy [Veloso, 1996] and earlier in the work on PLANIT [Drummond and Tate, 1992], an interactive planner's assistant.

The DSA approach also has several potential benefits in a stand alone setting. One aspect is in debugging a problem found in a planning result. As we said in Section 2.3.6.1, Chien identified two common problems resulting from knowledge encoding errors [Chien, 1996]:

- Incorrect plan generation
- Failure to generate a plan

In both instances, the DSA rationale can be used to quickly localise the error and fix the precondition, effect, or variable specification that may have caused the error. Domain additions and modifications can be reviewed as contributing to the plan space even if they weren't part of the "selected" plan solution. In many ways, the output of the DSA approach is similar to that provided by the UCPOP plan debugger (PDB) [Kwok, 1995] which we presented in Figure 2.3, page 51. The DSA Issues are like the PDB lines, the DSA options equate to PDB nodes, but notice that PDB doesn't support the presentation of the criteria or rationale for option selection.

[3] For example, the alternatives for Nonlin+DR's decision rationale is reflected by it's backward search state space, HTN, and plan space refinement methods.

4.5 Conclusion

In demanding, real-world planning situations we need "complete solutions" to address the associated requirements. Since planning can be viewed as a special type of design activity it makes sense to apply design rationale methods to planning as well. The design space approach views the solution as located in a space of possible elaborations. Capturing and externalising these elaborations creates a more robust solution that supports an intuitive inspection of the decisions made, the alternatives considered, and the influence of certain criteria on these alternatives.

The potential benefits of this approach were described for both a mixed-initiative and stand-alone AI planning settings. Outstanding items and issues have been raised to address more challenging settings. It was anticipated that the application of this approach to richer domains (e.g. Chapter 6)and more sophisticated planning situations would elicit a greater set of elements for a model of planning rationale.

Chapter 5

CPF Toolset

We presented the overall CPF architecture in Section 3.1.2. The tools described in this architecture have been implemented as the CPF toolset. We have, in fact, already discussed one component of this implementation, the CPM toolset (see Section 3.2.4) which supports the initial Requirements Analysis phase (see page 90). In this section we examine our implementation of the tool support for the other CPF phases as well. These tools rest on the solid representational foundation which we presented in Section 3.3 and utilize the rationale approach we developed in Chapter 4.

5.1 Toolset Properties

The generic CPF architecture outlined in Figure 3.2 on page 92 could be implemented in a variety of ways using any number of languages and computational platforms. In our implementation of this architecture for this thesis work, we chose to align this toolset with a few central, guiding properties.

Platform Independence We were very interested in creating a toolset that could be deployed on a wide range of platforms (e.g. Unix, Windows NT, Apple Macintosh).

Location Independence Secured access to process knowledge over a widely distributed network would allow process users to have access to process knowledge from any node on the network.

Integration Support We envisioned the possibility that these tools could be integrated with other process support tools in an organisation.

Specialisation It may be possible for the presentation of the tool constructs to be specialised for the particular platform of choice at runtime. In addition, the tools should support specialisation of the flexible constraint expressions as described in Section 3.3.2.12.

Our goal of platform independence was largely met by adopting Java [Flanagan, 1997] as the main development language in order to leverage the "write once, run anywhere" Java approach. The CPM toolset though was built on the HARDY meta case-tool which has implementations only for Windows and Unix platforms. Also, the CPA was built for Sicstus Prolog which we have only run on Unix but it should be able to be run on other Sicstus-compliant implementations on non-Unix platforms.

The idea of location independence is supported to some degree by our addition of FTP and TCP/IP based communication. Domain and process descriptions can either be accessed from a local filestore or written/read from an FTP file server. Connectivity to the CPA is enabled via a user-defined TCP/IP port communication channel.

Our view of integration support is that of an HTML page acting as a container for tool applets. Using Java, we can integrate the central process and domain editor panels with other tool panels as was demonstrated in the ACP3 Air Campaign Planning Process Panel approach for the DARPA/Air Force Research Laboratory (Rome) Planning Initiative (ARPI) TIE 97-1 in which an early version of CPE was used alongside a Course of Action (COA) evaluation matrix.

Finally, the specialisation property can be partially met for the domain and process editors by using the pluggable-look-and-feel API in Java 2. User definable plug-in software modules are also made possible for constraint expression builders by using the Java reflection API to load and execute customised class files at runtime.

5.2 Domain Editor

The Detailed Domain Development phase outlined in the CPF architecture (see page 90) is oriented toward the production of a detailed domain defin-

ition. In this thesis work we have provided an implementation of a tool which illustrates support for this phase. This section reviews that tool, the Common Domain Editor (CDE), and discusses its role in the CPF toolset.

5.2.1 Purpose

The purpose of the domain editor is to provide assistance in the development of a detailed domain definition. As the name of the output work product suggests, the focus of the tool is to facilitate the collection of details about the domain. These details provide the information necessary to formulate an operational representation of the domain which can be passed to external tools such as an AI planner (e.g. O-Plan). While it is possible that this detailed domain specification could simply be authored using a standard text editor, we believe that this graphical tool provides the appropriate level of interaction with the domain knowledge for organisational process users.

In the following sections we will discuss the user interface design and illustrate some of the options and support available during domain editing sessions. We will show that a domain editing session may begin with either a blank domain template, an initial domain specification (as output from the requirements methodology), or with an existing domain file from a previous editing session. In the tool summary we will reflect on the capabilities of this tool and describe how we have utilised it in our CPF implementation.

5.2.2 Interface Overview

The Common Domain Editor is implemented as a three panel configurable interface composed of the domain navigation panel, multi-process editing panel, and message panel. This is shown in Figure 5.1. We say that it is configurable because the proportion of interface space for any of the three panels may be adjusted at runtime from anywhere between 0 and 100%. This is accomplished by sliding the dividing panel boundaries or by using the arrow adjusters in the panel boundaries.

The domain navigation panel presents a structured collection of collapsible and expandable domain elements using a tree-based view. Each tree node has a type-specific menu which provides appropriate commands (e.g. add, edit, delete, properties,

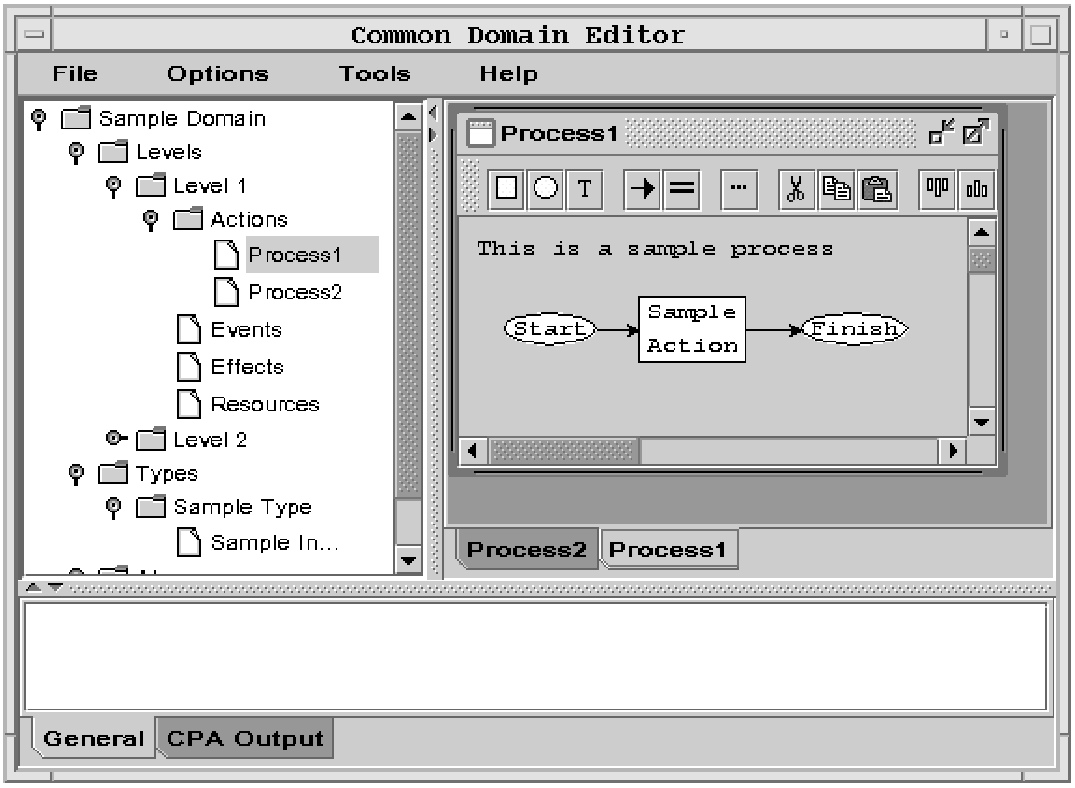

Figure 5.1: The Common Domain Editor interface

etc.). The root node of the tree represents the overall domain. Only one domain may be presented in this panel at a time (i.e. there can only be one root node). This node branches into three nodes representing the domain levels, types, and constraints which are attached at the domain model level (i.e. cpo-always-constraints).

The root domain **level node** encapsulates the collection of domain levels. Within each level are collections of the actions and events at that level and summaries of the aggregated level effects and resources. The action and event level entries act as pointers to processes which may be accessed in the multi-process editing panel which we will discuss in Section 5.2.3. Level ordering is significant as the levels are meant to be arranged in increasingly detailed perspectives where possible. Levels can be added above or below existing levels. Actions and events within a level can be cut and pasted from one level to another.

The root domain **types node** encapsulates the collection of activity relatable objects (AROs) for the domain. The nodes in this collection either specify a type (i.e. a new domain sort) or a type instance. The subsort relationship ("isa") can be graphically assigned between two types to form a subsort hierarchy. Type instances connected to types are meant to convey the sort-instance connection expressed via sort definitions which we discussed in Section 3.3.2.3.

Finally, the root "domain level" constraints are encapsulated by the **always node**. This is meant to convey the fact that these special constraints *always* apply for any process attached at any level in the domain. As with other constraints the actual expression of these constraints depends on the particular grammar associated with them. As we shall see, user-defined expression builders can be plugged in at runtime to provide custom constraint management.

The top-level menu for CDE is broken up into File, Options, Tools, and Help. The File menu provides access to dialog boxes for writing or reading domain specifications for either the local filestore or a named FTP server. The File menu also provides access to separate translation modules which we will discuss further in Section 5.2.4. The Options menu provides access to various visual customisations such as changing to platform-specific component presentations. The Tools menu provides access to external services such as the Common Process Assistant (CPA) which we will present in Section 5.4.

5.2.3 Multi-Process Editing Panel

The right-hand side panel in Figure 5.1 is a multi-process editing panel. The tabbed area shown below the panel can be used to switch focus between various domain process panes. Selected process panes can also be "brought to the top" by selecting their index entry in the domain navigation panel. Processes are displayed using a node-arc presentation. The rectangular nodes indicate actions (i.e. cpo-actions) and the oval nodes indicate subtypes of the "other node" type (e.g. cpo-start, cpo-finish). Conceptually, action nodes have two "halves" (one from the left edge to the center and from the center to the right edge) which represent the begin and end timepoint pairing (see Section 3.3.2.7) whereas "other nodes" only associate with one timepoint.

Arcs may be dragged from source nodes to targets to graphically assign temporal relationships between the underlying timepoints. When originating or releasing a "drag" the pointer position on the action node is used to determine whether the constraint applies to its left half (begin timepoint) or right half (end timepoint). Arc types may either be directed single lines or undirected double lines which equate to a before or equal temporal constraint expression respectively (see Section 3.3.2.15).

Nodes may be selected in various ways (e.g. dragging out a bounding selection box, clicking on a single node, etc.). Selected nodes can be dragged around or aligned with commands to position them on the scrollable process canvas. Most operations on nodes can either be carried out using context sensitive popup menus, the dockable toolbar, or by direct mouse manipulation.

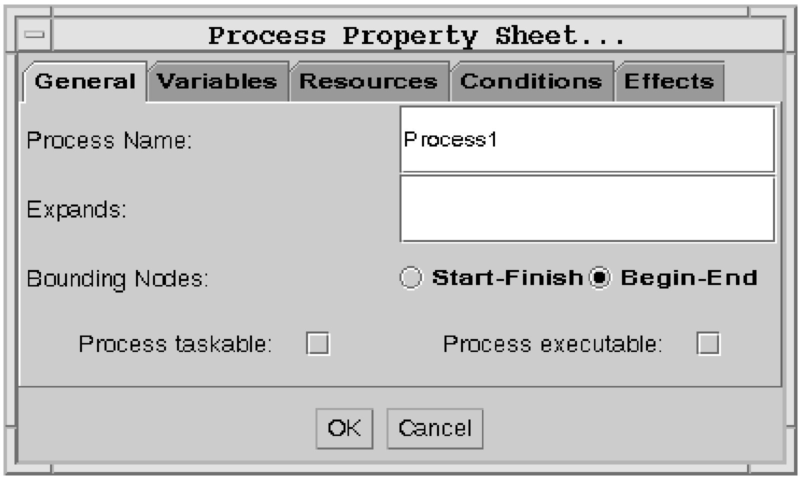

Figure 5.2: Process properties dialog

Both the containing process and the contained nodes have "property dialogs" associated with them as shown in Figure 5.2. The property dialogs provide interfaces for attaching and managing detailed process constraints. For example, there are tabs for variable (cpo-variable-constraints), resource (cpo-resource-constraints), condition (cpo-input-constraints) and effect (cpo-output-constraints) constraints. These interfaces simply manage lists of expressions, the content of which is determined by the grammar specified for a particular application. Plug-in expression builders can be loaded at runtime to help build custom expressions.

Textual annotations can be attached and positioned similar to action or other nodes (e.g. the "This is a sample process" string in Figure 5.1). These are used for attaching unstructured strings to the process specification (i.e. cpo-annotation-constraints). Rationale for domain process design can be captured in a separate rationale window within a process pane. We will delay discussing this though until we get to Section 5.3.2.

5.2.4 Domain Editing

The main tasks involved in detailed domain editing revolve around cycles of loading, modifying, and saving domain knowledge. In the normal workflow of the CPF architecture (see Figure 3.2) this begins with the initial domain specification which was translated from the combined thread diagrams (CTD) generated in the CPM toolset. This process may also begin with a blank domain shell as well.

During the loading of an existing domain specification, the CDE utilises a robust, custom parser module generated by SunTest's Java Compiler Complier (JavaCC). This freely available tool takes a grammar specification (in this case the CPL grammar, see Appendix B) and converts it to a Java program that can recognise matches to the grammar. Very detailed parsing errors are presented to the user in the message panel.

The domain editing session may either be run as an applet within an HTML page or as a stand alone Java program. In order to run it as an applet though certain security measures must be put in place to allow the applet to work beyond the default applet sandbox security model[1].

Once satisfactorily completed, domain definitions may then be translated to target languages for use outside the framework. An example translation we have implemented is a one way translation from a CPD specification to the O-Plan Task Formalism Version 2.3 [Tate *et al.*, 1994a]. We will discuss this translation in more detail in Section 5.5.

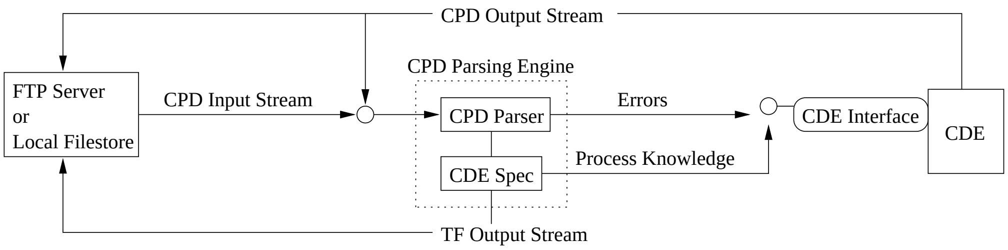

Figure 5.3: The decoupled CPD parsing engine

The parser module we mentioned above (which again is largely automatically gener-

[1] See the CDE tool documentation for information on setting up public/private keys, digital signing, and policy files.

ated using JavaCC) is actually tightly coupled with a CDE specification module as we illustrate in Figure 5.3. This module pairing makes up the CPD parsing engine which is decoupled from the rest of the CDE application. The specification module acts as an intermediate representation store which is built up as the parser goes about its job of recognising the grammar. The specification module knows how to either build processes in CDE by invoking the CDE interface methods or to translate the specification to TF. The translation service may be accessed in one of two ways: via the CDE File command menus or at the command line. Invoking at the command line (i.e. without having to load and run the CDE application) allows a user to stream in CPD input from a filestore and to produce a TF output stream. Invoking translation from within the CDE application begins by simulating a file save but then redirects the CPD output stream (in memory) through the same channel used by the command line method.

5.2.5 Summary

This implementation of the domain editor component which was outlined in the CPF architecture serves as a "proof of concept" tool supporting the detailed domain development phase. It illustrates the level of interaction which we believe is appropriate to help organisations synthesise and manage knowledge of the processes in their particular domain. The tool essentially provides a presentation of the underlying knowledge content which again is rooted in an AI planning-based representation. It can be customised to support the flexibility which is built into the shared framework interlingua. The tool demonstrates the interoperability required in the framework by integrating with target-specific translators.

5.3 Process Editor

The Process Management phase outlined in the CPF architecture (see page 90) is oriented toward the visualisation, maintenance, and communication of synthesised process knowledge. In this thesis work we have provided an implementation of a tool which illustrates support for this phase. This section reviews that tool, the Common Process Editor (CPE), and discusses its role in the CPF toolset.

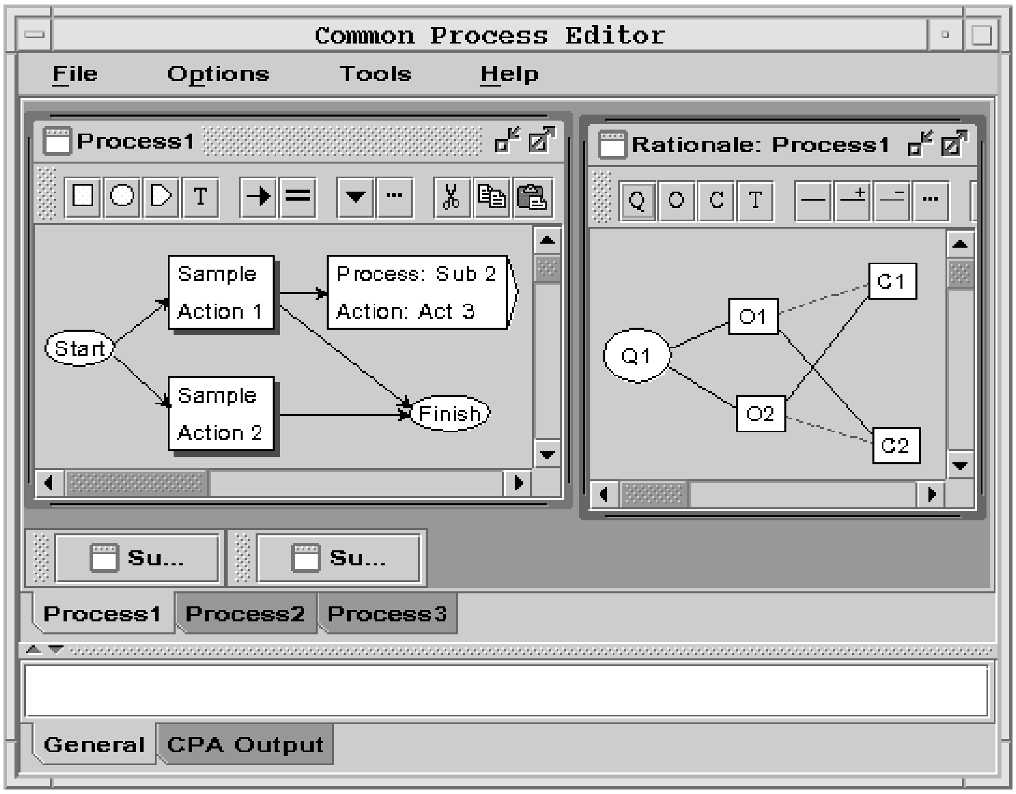

Figure 5.4: The Common Process Editor interface

5.3.1 Purpose

The main purpose of the process editor is to assist in the management of new processes synthesised from the domain knowledge elicited in the domain editor. The focus of the tool is to facilitate visualisation and modification of this knowledge. The process knowledge may be enriched either through direct user interactions (e.g. adding additional constraints) or by software integration exchanges with other tools. The latter exchange of knowledge is supported by the translation aspects of CPF. For example, it may be necessary to translate process knowledge into a native format to be used with a process evaluation tool which will then provide a qualitative or quantitative assessment of the process. It is assumed that such an exchange is practically possible and is discussed in Section 5.5.

As with the domain knowledge in CDE, it is possible that this information could be simply presented within a standard text editor, but we believe that this graphical tool provides the appropriate level of interaction with the process knowledge for organisational process users. In the following sections we will discuss the user interface design and illustrate some of the options and support available during process editing sessions. We will show that a process editing session may begin with either a blank process template, an initial process description (as output from an AI planner), or with an existing process file from a process repository. In the tool summary we will reflect on the capabilities of this tool and describe how we have utilised it in our CPF implementation.

5.3.2 Interface Overview

The process editor interface (see Figure 5.4) is very similar to the domain editor interface discussed in Section 5.2.2. The obvious difference is the lack of a domain navigation panel. The multi-process editing panel and the message panels are both reused in this tool. The CPE multi-process editing panel is slightly different than the one we presented in Section 5.2.3 though.

As opposed to processes in CDE, processes manipulated in CPE may have some assigned hierarchical structure (see Section 2.3.1.3 on hierarchical task networks). That is to say that a particular decomposition of an action may have been defined within the overall process description. We illustrate this graphically by providing a shadowed effect[2] around a process node. This is shown in Figure 5.4 for "Sample Actions 1 and 2". CPE presents a node's expansion in a separate window (currently minimised on the multi-process panel). Toolbar and context-sensitive menu commands permit navigation through the task network decompositions.

As a side effect of cleanly separating the presentation of decomposed sub-processes there is a need to provide access across process window boundaries. For example, consider Figure 5.5. This diagram essentially combines the information from the three process windows from Figure 5.4's Process1. The dashed line in Figure 5.5 represents an ordering constraint which spans two windows in the CPE presentation (Process1 and Sub2). In order to provide a way to represent this while still maintaining our process-per-window user interface guideline we introduce a new node component, the "node reference". A node reference creates a proxy representation of a node existing

[2] This is similar to what is found in other graphical process presentation notations such as IDEF3 [Mayer *et al.*, 1992].

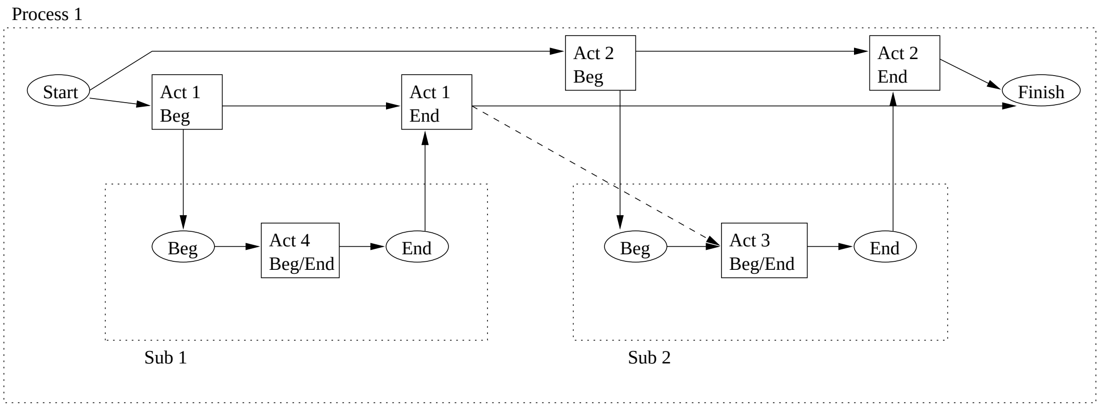

Figure 5.5: Cross window ordering requirement

in another process in order to allow orderings to be made within the window but which in fact extend over these window boundaries. This node reference presentation is illustrated in Figure 5.4.

We can also see from Figure 5.4 how the user has the ability to visualise the process' decision rationale in a separate window on the multi-process editing panel. The underlying representation is based on the rationale extension which we discussed in Section 3.3.2.20. The Questions, Options, and Criteria, along with the relations assigned between them are displayed as a design space. This rationale can be maintained alongside the process structure as further editing and process management activities are conducted throughout the lifecycle of the process. See Figure 4.1 for an explanation of this graphical rationale presentation.

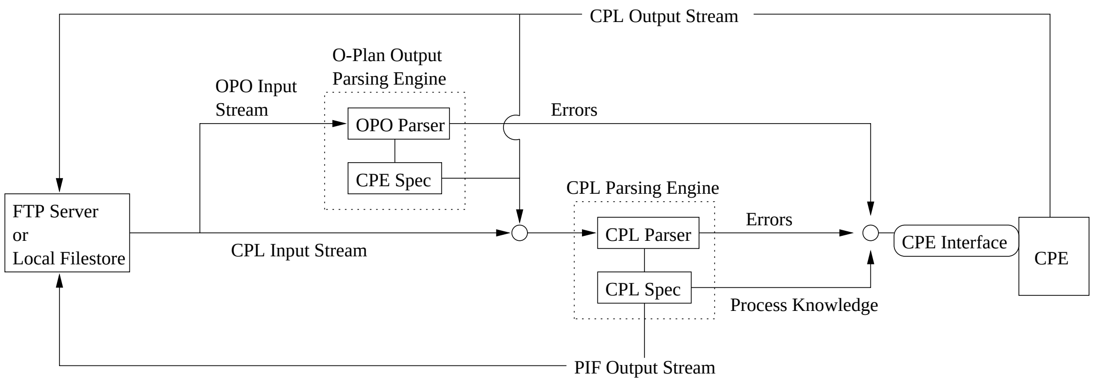

Figure 5.6: The decoupled OPO and CPL parsing engines

5.3.3 Process Editing

The main tasks involved in process management revolve around cycles of loading, modifying, and saving process description knowledge. In the normal workflow of the CPF architecture (see Figure 3.2) this may begin with the output of a plan from an AI planning system. During our work with the O-Plan planning system we determined a need to enrich one of its standard modes of communicating plan knowledge in order to evaluate our approach. Using O-Plan, a user can output very limited information about a synthesised plan to a file[3]. This information is expressed in the O-Plan Output Format (OPO) which is specified in the Task Formalism manual [Tate *et al.*, 1994a]. This expression only contains a listing of the nodes and temporal relationships between them and omits useful knowledge, such as activity effects, dependencies, and resource usage. Our modification to this output module adds this functionality and can be added to an O-Plan source directory.

The process editing session may either be run as an applet within an HTML page or as a stand alone Java program. As we can tell from the CPF architecture (Figure 3.2), an implementation of the process editor should be prepared to take input from the standard process store representation (i.e. CPL, see Section 3.3.2). We can also see that there are translation steps from and to the process editor in order to provide the interoperability we have described in the framework. As with the CDE, we have decoupled the parsing engines from the main CPE implementation which we illustrate in Figure 5.6.

Robust, custom parsing is again provided by a parser module generated by SunTest's publicly available JavaCC tool based on our definitions of the extended O-Plan output and CPL grammars. In Figure 5.6 we can see that the OPO parsing engine and the CPL parsing engine act in serial. Both engines have the two part split we first mentioned in Section 5.2.4. In the case of OPO, we build a CPE specification as the input OPO grammar is recognised. Once completed, that specification is streamed into the same CPL parsing engine which is used to read CPL specifications from file. The CPL specification module encapsulates the behaviour to either load a process description into

[3] There are other ways to link into O-Plan to obtain more detailed plan knowledge such as through the plan world viewer interface [Tate and Drabble, 1995].

CPE or to translate it to another interchange format or hub language (see Figure 3.3). As this is completely decoupled, users may choose to run the CPL or OPO parsing engines from a command line or from within the CPE user interface.

5.3.4 Summary

This implementation of the process editor component which was outlined in the CPF architecture serves as a "proof of concept" tool supporting the process management phase. It illustrates the level of interaction which we believe is appropriate to help organisations visualise and maintain knowledge of newly synthesised processes in their particular domain. The tool essentially provides a presentation of the underlying knowledge content which again is rooted in an AI planning-based representation. It can be customised to support the flexibility which is built into the shared framework interlingua. The tool demonstrates the interoperability required in the framework by integrating with target-specific translators.

5.4 Process Assistant

In this section we present an implementation of a CPF analysis tool which we described in the CPF architecture (see page 92). The analysis tools illustrate knowledge-based components which can be accessed by either the domain or process editor in order to provide support for managing rich organisational process knowledge. The focus of this particular tool is restricted to providing an interval-based evaluation of a process description using a mapping from its timepoint-based expressions. Errors, explanations, and possible resolutions are communicated back to CPF toolset users.

The Common Process Assistant (CPA) is a Prolog-based tool for analysing process knowledge. It is an example of one of the possible implementations of a CPF analysis tool. Currently, the tool can be used to evaluate temporal relationships of a process and to provide information on errors, explanations of those errors, and limited suggestions to resolve them. In this section we will briefly consider the applicability of timepoint and interval theories with respect to their use in a pragmatic framework such as CPF. In doing so, we will summarize our research into mapping from timepoints to intervals which will look in depth at the various possible configurations which might be encountered during this mapping. We will show what use this knowledge of various configurations can add to tools such as CPE and CDE.

5.4.1 Representations of Time

In this thesis work on a Common Process Framework we have sought to define a lifecycle of activities surrounding the management of process knowledge. As we have seen, our approach is centred around knowledge-rich processes which are based on past and present work in Artificial Intelligence planning and plan representations. In addition to this, we have also gained insight through our involvement with standards work related to the exchange of process knowledge.

At the heart of this framework is a process ontology, the Common Process Ontology (CPO), which we presented in Section 3.3.1. The ontology defines terms and concepts which can be used to describe the space of process behavior from a design-based perspective. As we have shown CPO is based on a constraint model of activity (viz. [Tate, 1995, Tate, 1996c]) in which processes may be designed by adding various types of constraints including temporal, resource, spatial, etc.

In the design of an ontology there are typically several choices in representing and defining various concepts. An ontology describing plans or processes is certainly no exception. One of the most important ontological design decisions in developing such an ontology is its formalisation of time. Various levels of commitment can be made to the structure of time. As a general "rule of thumb" though we note that high levels of commitment restrict the applicability of the ontology while lower levels of commitment may offer general terms and definitions which can be specialized as needed.

A fundamental issue though involving an ontological commitment to time revolves around a basic stance on what we *mean* by "time". At least two of the possible senses embody either **timepoints** or **time intervals**. The following excerpts from [Hayes, 1996] outlines these two approaches. Regarding intervals

> "[They are] ... pieces of time; physical entities whose sole dimension is time-dimension. These are variously called time-periods or time-intervals,

> or simply intervals. Examples include during the 1994 winter Olympics, the sixteenth century and 10:50 to 11:00 a.m. on 30 May 1993. These are particular pieces of time located in (or perhaps, parts of) the time-plenum. Intervals are in many ways the most central concept for temporal reasoning since they are the temporal extents of things. Events typically are thought of as occupying them, propositions are true during them and they are the lifetimes of objects."

A timepoint-based approach on the other hand is distinguished as well

> "[Another] notion is that of a timepoint. Exactly what counts as a point, and the relationships between points and intervals, seem to be particularly controversial and sensitive questions, and many of the formalisations in use in computer science have taken one or another stance on the answers to these questions."

There are times when we may prefer to work with timepoints and there are other times when time intervals are more appropriate. For example, in specifying a process we may wish to have a highly flexible approach in which we can attach begin and end timepoints to activities and then order those timepoints using "before" or "equals" relations. As a result of evaluating such a network though we may wish to explain errors in a particular specification designed with such constraints from an interval perspective rather than a point-based perspective (e.g. we would expect something like "activity A cannot be after activity B given that etc." rather than "the end point of activity B cannot be ordered before the begin point of activity A given that etc.").

In fact, this particular scenario is enacted within the CPF. The java-based domain and process editors (Sections 5.2 and 5.3) permit interaction with node ends of activities which are associated with timepoints. When the user chooses to evaluate the process network, it is sent to a Prolog-based Common Process Assistant (CPA) via TCP/IP. See Figure 5.7 which shows the CPE/CDE interface to CPA where users configure the CPA location, select a process to be evaluated, and execute the analysis[4]. The

[4] The CPA can listen on any port and can be placed on any machine accessible to the editors over a network and which has a copy of Sicstus Prolog.

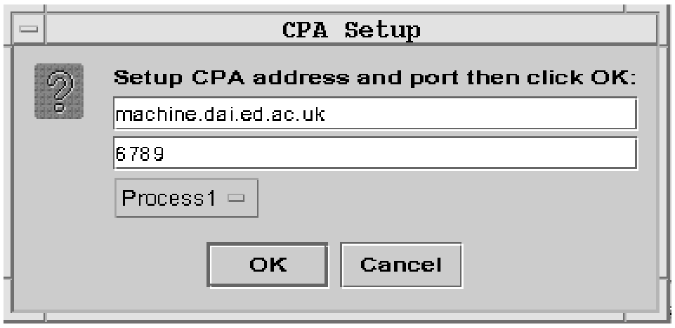

Figure 5.7: Accessing the CPA analysis tool

CPA attempts to map this timepoint-based knowledge into interval-based knowledge so that it can use interval-based reasoning to detect illegal configurations. The CPA then reports back errors, rationale, and suggestions from an interval-based perspective. For example, Figure 5.7 shows the output from a network analysis in which one of the transitive interval relationships has been violated.

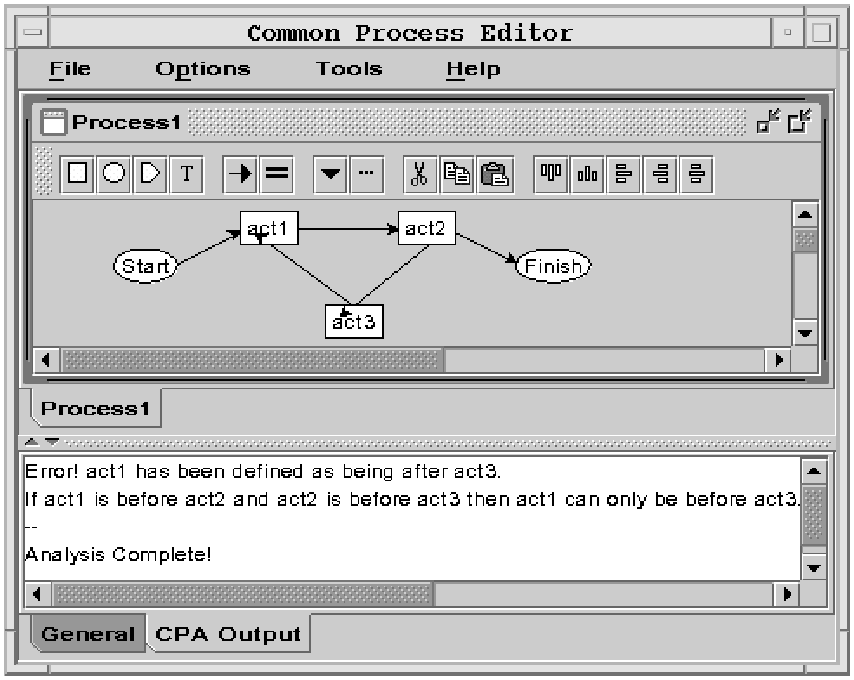

Figure 5.8: Errors and explanations in CPA

It is the intent of the next several sections to explore the CPA mapping of the actions and their respective timepoints onto time intervals and to illustrate how this knowledge of the mapping can be used to prevent illegal or unnecessary constraints in the editors. Following this we will discuss how CPA detects errors and provides temporal advice.

5.4.2 CPO to CPA

A timepoint in CPO, $\mathcal{T}p$, characterises a specific, instantaneous point that lies along a line which is an infinite sequence of time points. Conceptually, pairs of timepoints for nodes and processes delimit a time interval. So given that, how do we map sets of temporal constraints on timepoints into interval relationships? First, we must enumerate the possible set of interval relationships that we will be interested in. For this, we turn to Allen's definition of the 13 relations between intervals [Allen, 1984a, Allen, 1984b] which is typically taken to be the standard set. This set of relationships includes the following six **{before, meets, overlaps, starts, during, finishes}** along with their inverses and **equals**.

We can utilise an axiomatisation based on Hayes' catalogue of temporal theories [Hayes, 1996] in order to map timepoints and ordering constraints into these 13 relationships. In fact, during our application of these definitions, we spotted and corrected a couple of errors in the source mapping axiomatisation. The correct axioms are listed in Appendix entry C.1.

As we have indicated, this axiomatisation is used in the Common Process Assistant (CPA) to map the timepoints and ordering constraints which are passed from the process and domain editing tools, upon a users request, into an interval theory for consistency checking. Allen's table of legal relationships between intervals [Allen, 1984a] is then used to detect errors and to provide rationale for why a process specification is incorrect (i.e. CPA explains which legal interval relationships could exist). For example, Figure 5.8 shows a simple Common Process Editor specification which was passed to the CPA. Recall that, in CPE, the left half of the node is used to graphically indicate the begin timepoint while constraint attachments to the right half indicate relationships assigned to its end timepoint.

The CPA output message area displays the information passed back from CPA which includes the set of errors along with an explanation and suggestion of possible corrections for each error. In this case there is only one error. It should be noted that the nature of this reported error depends on the order in which the constraints are processed. For example, the error could also have been reported to be between act1 and act2 or act2 and act3. When searching for errors, CPA maintains a set of known errors which are a triple, <i,j,k>, in which the relationships between intervals i and j and j and k conflicts with a relationship between k and i. When it realizes that there is an error already reported involving the intervals i, j, and k it refrains from repeating the error using different bindings.

5.4.3 Analysis: Timepoints and Intervals

In this section, we explore the relationship between timepoints and time intervals in a bit more detail. In particular, we are interested in the implications of our approach whereby temporal constraints on timepoints are incrementally added to an activity specification and how that affects a mapping to time interval relationships between 2 activities. Two important perspectives we address are: the validity of a set of temporal constraints; and the cases in which those constraints completely or partially specifies some Allen relationship between the intervals implied by the activities. For this analysis we will outline some basic terms and notations.

Each activity, $\mathcal{A}$, has 2 timepoints which we will abbreviate as: $\mathcal{T}p^{\mathcal{A}}_{begin}$ and $\mathcal{T}p^{\mathcal{A}}_{end}$. There is one relation that always exists between an activity, A_1, timepoint pair: $before\_tp(Tp^{A_1}_{begin}, Tp^{A_1}_{end})$. No other relation can be made between these two points. For any two different activities in a process activity specification, $\mathcal{A}s$, there is a set of unique pairs of timepoints, which we will refer to as a $\mathcal{T}set^{\mathcal{A},\mathcal{A}}$, which is defined

$$Tset^{A_1,A_2} = \{(tp_1, tp_2) | tp_1 \in \{Tp^{A_1}_{begin}, Tp^{A_1}_{end}\} \wedge tp_2 \in \{Tp^{A_2}_{begin}, Tp^{A_2}_{end}\}\}$$

Each pair in a $\mathcal{T}set$ may be related in one of two possible ways or not at all. If a relationship is assigned to a pair then either one timepoint is temporally before the other or they are equal. We will express this in an infix notation as

$$\begin{aligned} tp_1 \prec tp_2 &\equiv before\_tp(tp_1, tp_2) \\ tp_1 = tp_2 &\equiv equal\_tp(tp_1, tp_2) \end{aligned}$$

Given this information, we can see that there are 256 unique configurations of relationships between this set of pairs. This calculation is arrived at in the following way

$$\begin{array}{rl}
1 & 0-assignment \\
+12 & 1-assignment \\
+(12*9)/2 & 2-assignment \\
+(12*9*6)/6 & 3-assignment \\
\underline{+(12*9*6*3)/24} & 4-assignment \\
256 & unique-combinations
\end{array}$$

For this calculation, the space of combinations is divided into five sets: the set containting 0 user-specified constraints between the timepoints (0-assignment); the set containing 1 user-specified constraint (1-assignment); 2 (2-assignment); etc. up to four constraints. Note that once a single constraint between two timepoints has been selected, it reduces the possible set by 3, not 1. For example, if a user specifies that $(Tp^{A_1}_{begin} \prec Tp^{A_2}_{begin})$ then it also eliminates $(Tp^{A_2}_{begin} \prec Tp^{A_1}_{begin})$ and $(Tp^{A_1}_{begin} = Tp^{A_2}_{begin})$ as well.

Of course these are not all legal combinations of constraints. An example of an illegal 2-assignment combination would be $\{(Tp^{A_2}_{end} \prec Tp^{A_1}_{begin}), (Tp^{A_1}_{begin} \prec Tp^{A_2}_{begin})\}$. Clearly, if A_2 is before A_1, which is what the first constraint says, then it cannot be the case that $Tp^{A_1}_{begin}$ is before $Tp^{A_2}_{begin}$.

5.4.3.1 0-assignment

We could start by looking at these constraints level by level. In the first level (0-assignment) there is only one possible configuration which corresponds to no user-specified assignments being made between any of the $\mathcal{T}set$ elements. Note that we say user-specified, because there are two implied constraints always present: $\{(Tp^{A_1}_{begin} \prec Tp^{A_1}_{end}), (Tp^{A_2}_{begin} \prec Tp^{A_2}_{end})\}$ which comes from the CPO ontology.

5.4.3.2 1-assignment

Stepping down a level, we can see that there are twelve unique, single assignment configurations between any $(tp_1, tp_2) \in Tset^{A_1,A_2}$. This is very straight forward as we can see that there are a total of 12 unique constraints which could be assigned. These include

$$\begin{array}{rcl}
\{(Tp^{A_1}_{begin} \prec Tp^{A_2}_{begin})\} & \vee & \{(Tp^{A_1}_{begin} \prec Tp^{A_2}_{end})\} \vee \\
\{(Tp^{A_1}_{end} \prec Tp^{A_2}_{begin})\} & \vee & \{(Tp^{A_1}_{end} \prec Tp^{A_2}_{end})\} \vee \\
\{(Tp^{A_2}_{begin} \prec Tp^{A_1}_{begin})\} & \vee & \{(Tp^{A_2}_{begin} \prec Tp^{A_1}_{end})\} \vee \\
\{(Tp^{A_2}_{end} \prec Tp^{A_1}_{end})\} & \vee & \{(Tp^{A_2}_{end} \prec Tp^{A_1}_{begin})\} \vee \\
\{(Tp^{A_1}_{begin} = Tp^{A_2}_{begin})\} & \vee & \{(Tp^{A_1}_{begin} = Tp^{A_2}_{end})\} \vee \\
\{(Tp^{A_2}_{begin} = Tp^{A_1}_{end})\} & \vee & \{(Tp^{A_2}_{end} = Tp^{A_1}_{end})\}
\end{array}$$

In Appendix entry C.2 we provide tables and figures which outline our results of the analysis at each level (> 0). For example, Table C.1 lists the 12 1-assignment configurations. The following substitutions have been made to make the table more legible $B_1 = Tp^{A_1}_{begin}$, $E_1 = Tp^{A_1}_{end}$, $B_2 = Tp^{A_2}_{begin}$, $E_2 = Tp^{A_2}_{end}$. The entries in the right-most column index into Figure C.1. This figure graphically displays these unique, legal configurations and separates them into partial and complete sets. We note that all 12 possible configurations are legal and 4 configurations completely describe an Allen relationship between the intervals and the remaining 8 do not, which we call "partial" configurations.

Before we move on to explore the **2-assignment**, **3-assignment**, and **4-assignment** sets we will provide a generic description of the procedure we are following to analyze this space of possible configurations.

5.4.4 Analysis Procedure

The following algorithm defines the procedural steps we are following to inspect the space of possible timepoint-based constraint configurations[5]. In this algorithm we are

[5] Note that this is describing our analysis of the mappings, this isn't the error analysis procedure used in CPA.

interested in deriving a count of legal specifications (ls), partial specifications (ps), complete specifications (cs), and the list of legal, unique constraint sets (ucs).

```
1  PROCEDURE analyse-configurations()
2  BEGIN
3      ls ← 1;
4      ps ← 1;
5      cs ← 0;
6      ccs ← {(Tp_{begin}^{A_1} ≺ Tp_{end}^{A_1}), (Tp_{begin}^{A_2} ≺ Tp_{end}^{A_2})};
7      pcs ← possible-constraint-set();
8      ucs ←
9      {{(Tp_{begin}^{A_1} ≺ Tp_{end}^{A_1}),
10     (Tp_{begin}^{A_2} ≺ Tp_{end}^{A_2})}};
11     analyse-subroutine(ccs);
12     PROCEDURE analyse-subroutine (ccs′)
13     BEGIN
14         FOR c ← EACH ELEMENT OF pcs
15         DO BEGIN
16             IF (legal({c ∪ ccs′})
17             AND {c ∪ ccs′} ∉ ucs)
18             THEN BEGIN
19             ls ← ls + 1;
20             ucs ← {c ∪ ccs′} ∪ ucs;
21             IF (complete({c ∪ ccs′})) cs ← cs + 1;
22             ELSE ps ← ps + 1;
23             IF (count({c ∪ ccs′}) < 4)
24                 analyse-subroutine({c ∪ ccs′});
25             END;
26         END;
27     END;
28 END;
```

In lines 3-10 we initialise a number of variables to setup the call to the recursive analyse-subroutine procedure. Note that this procedure is defined in a similiar way to a standard ML "local function". By this we mean to indicate that the procedure is not restricted to using parameters and local variables, it may freely access variables that are defined within its scope.

The legal specification counter (ls) and partial specification counter (ps) are both initialized to 1 to represent the 0-configuration we discussed above. It is obvious then that there are no complete specifications (cs=0) at this stage (0-assignment). We initialize the current constraint set (ccs) to include those implied by the CPO ontology. The possible constraint set (pcs) is an unchanging set which contains the 12 we introduced above:

29 FUNCTION possible-constraint-set (): SET
30 BEGIN
31 return(
32 $\{(Tp^{A_1}_{begin} \prec Tp^{A_2}_{begin}), (Tp^{A_1}_{begin} \prec Tp^{A_2}_{end}),$
33 $(Tp^{A_1}_{end} \prec Tp^{A_2}_{begin}), (Tp^{A_1}_{end} \prec Tp^{A_2}_{end}),$
34 $(Tp^{A_2}_{begin} \prec Tp^{A_1}_{begin}), (Tp^{A_2}_{begin} \prec Tp^{A_1}_{end}),$
35 $(Tp^{A_2}_{end} \prec Tp^{A_1}_{end}), (Tp^{A_2}_{end} \prec Tp^{A_1}_{begin}),$
36 $(Tp^{A_1}_{begin} = Tp^{A_2}_{begin}), (Tp^{A_1}_{begin} = Tp^{A_2}_{end}),$
37 $(Tp^{A_2}_{begin} = Tp^{A_1}_{end}), (Tp^{A_2}_{end} = Tp^{A_1}_{end})\});$
38 END;

Finally, the list of unique constraint sets (ucs) is initialised to include one element, the single configuration in the 0-assignment level. Note that while ccs may be assigned a set of constraints, ucs is actually a set of sets of constraints.

The local analyse-subroutine procedure accepts the current constraint set as a parameter (line 12). For each possible constraint, the algorithm determines if the union of the selected constraint and the current constraint set is legal (i.e. the set of constraints do not lead to a contradiction and there are no repeats). In addition to this, the ucs is consulted to see if this proposed configuration has already been identified. If the proposed set is legal and unique the counter is incremented and the configuration is added into the ucs.

It is then evaluated to see if it completely specifies an Allen relationship. This can be determined by using the axiomatization in the Appendix. The appropriate counter is then incremented (lines 21 or 22). Finally, if the number of constraints in this new configuration does not equal the maximum (4) then we must evaluate the possibility of adding additional constraints. The analyse-subroutine is called recursively with the new constraint set. When the analyse-configurations procedure is complete, we should have all of the desired information acquired at line 28.

5.4.4.1 1-assignment (revisited)

We observed that there were 12 unique, legal configurations for the 1-assignment set. Four of those 12 "completely specify" an Allen interval relationship (along with the inverses)., viz.

$$\begin{aligned}
Tp^{A_1}_{end} \prec Tp^{A_2}_{begin} &\equiv before(A_1, A_2) \\
Tp^{A_2}_{end} \prec Tp^{A_1}_{begin} &\equiv after(A_1, A_2) \\
Tp^{A_1}_{end} \prec Tp^{A_2}_{begin} &\equiv meets(A_1, A_2) \\
Tp^{A_2}_{end} \prec Tp^{A_1}_{begin} &\equiv met-by(A_1, A_2)
\end{aligned}$$

The remaining 8 1-assignment configurations can be considered to specify "partial" Allen interval relationships, that is to say that they cannot yet map into any of the identified 13 relationships given the axiomatization. As we mentioned before, this is summarized in Table C.2 and Figure C.2.

5.4.4.2 2-assignment

Table C.2 summarizes the validity of the relationship between C_1 and Figure C.1 for the 2-assignment level. By C_1 mean some constraint which was selected by line 14. The x-axis of Table C.2 is populated by the set of legal specifications which were derived from the previous level (i.e. 1-assignment in this case). Since this is a reflexive relation only half of the table needs to be generated. Legal pairs of assignments are referenced by the number which indexes into the graphical presentation in Figure C.2. In addition to this, we have placed a '$\times$' to indicate points at which the test at line 16 failed. Failure of the test at line 16 is also indicated by '$\bullet$' which means that the proposed constraint was a repeat of one already in the current constraint set. As you can see from Table C.1 there are configurations which map into the same, unique specification (i.e. they share the same index). This is detected at line 17.

We can see from Figure C.2 that there are actually 18 unique, legal 2-assignment configurations. Ten of them are "complete" Allen interval relationships and the remaining 8 are still "partial". At this level, only the *overlaps* and its inverse *overlapped* $-$ *by* relationship from Allen's set of 13 cannot be expressed. These remaining relationships require at least a 3-assignment configuration.

5.4.4.3 [3,4]-assignment

At 3-assignment, we have 14 configurations which are "complete" Allen interval relations and 5 are "partial" for a total of 19 unique, legal 3-assignment specifications (see Table C.3 and Figure C.3). Note that while the y-axis of the tables remains the same, the x-axis depends on the legal specifications uncovered in the level above it. In this case the Figure C.2 specifications.

Finally, at 4-assignment we have 7 "complete" and 0 "partial" configurations (Table C.4 and Figure C.4) corresponding to all of the 13 interval relationships (given the inverses).

5.4.5 Mapping Results and Discussion

As we can see, only 57 of the 256 unique combinations are legal ($\approx 22\%$) and only 35 of these 57 completely specifies an Allen interval relationship ($\approx 14\%$ of the total). This analysis provides knowledge we can use to construct efficient process editors which prevent users from specifying illegal configurations between two activities (while relying on the CPA interval reasoning to check transitive relationships). Consider the case where some user-defined constraint, C_1, already exists between a pair $(tp_1, tp_2) \in Tset^{A_1,A_2}$. Given this constraint between (tp_1, tp_2) (or given some set of constraints) we can encode a table lookup which tells us what other constraint, C_2, can be assigned to a pair, $(tp_3, tp_4) \in Tset^{A_1,A_2}$, whereby C_2 is still consistent with the assertion from C_1.

In addition to this, it is obvious that some of the configurations contain superfluous constraints which can be safely eliminated. For example, the $Tp^{A_j}_{end} \prec Tp^{A_i}_{end}$ constraint in the specification labelled "17" in Figure C.2 is unnecessary as we can infer this relation given that $(Tp^{A_i}_{begin} = Tp^{A_j}_{end}) \wedge (\mathcal{T}p^{\mathcal{A}}_{begin} \prec \mathcal{T}p^{\mathcal{A}}_{end})$.

5.4.6 Errors and Explanations

Up to this point in the overall CPA section we have mainly focused on the ramifications of mapping from timepoint-based constraints into Allen's interval calculus. While we cited that it is more "natural" to speak of errors in a process description from an interval perspective, we actually have a more practical reason why we would like to recast this representation using the interval calculus. In particular we would like to draw on the knowledge contained in Allen's transitivity table of legal relationships which can exist between intervals [Allen, 1984b].

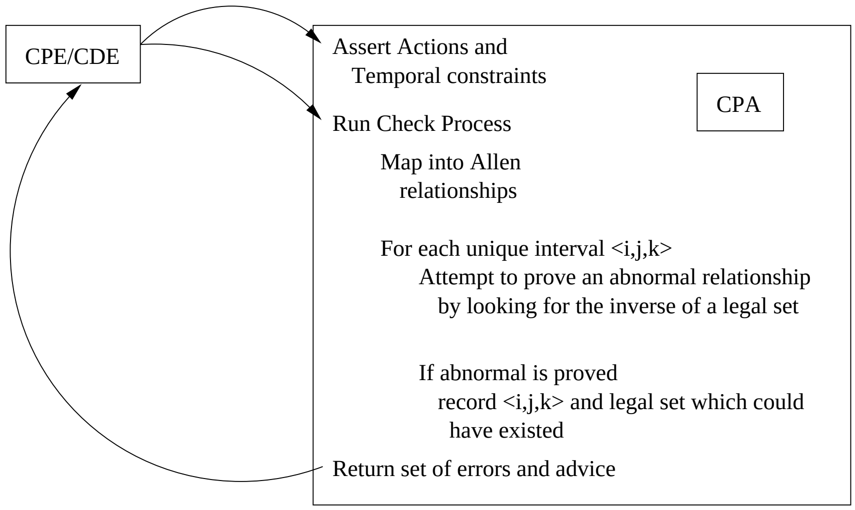

Figure 5.9: CPA method to scan for temporal errors

The transitivity table can be used in the following way. Assume that we have intervals A, B and C. Let's say that we know which of the thirteen relationships exists between A and B and which relationship exists between B and C. A lookup in the table will then tell us which relationships can "legally" exist between A and C. So let's look at the example we provided in Figure 5.8. Let A be the interval defined by act1 and B be the interval defined by act2 and C be the interval defined by act3. A maps into the "before" Allen relationship with respect to interval B. B maps into the "before" Allen relationship with respect to interval C. So our question is what legal relationship can exist between A and C? Looking it up in the table we discover that C can only be after A. Unfortunately though in our process description we have specified that C is before A which results in an error along with the appropriate advice.

This overall process is summarised in Figure 5.9. In CDE or CPE a user can specify a process description. When the user requests a process analysis (see Figure 5.7) the tool first sends over its knowledge of actions and temporal constraints (cpo-actions and cpo-ordering-constraints). Next, CPE or CDE signals the start of the check process. As we have been discussing throughout this section, we first map this knowledge into complete Allen relationships. In CPA we have encoded the transitivity table as a set of Prolog clauses which we use to prove "abnormal" relationships. An "abnormal" relationship is basically a triple which violates the known set of legal transitive relationships. Armed with this knowledge, we can not only flag the error but also tell the user which transitive relationships are legal alternatives that could be specified to correct the error.

5.5 Process Translators

This final section of the CPF toolset discusses the implementation of the various translation modules (CPTs) used to establish the interoperability between the Common Process Language (CPL) and other application or environment-specific representations (see Figure 3.3). We discuss some of the basic mappings and issues involved in tackling this integration and knowledge exchange.

In this section we describe the implementation of the translation modules (CPTs) which we have designed as part of the CPF toolset. We can distinguish various types of translation modules based on the mechanism by which knowledge is acquired and distributed. This is depicted in Figure 5.10. Some translation modules work by reading knowledge from some defined interface which is exposed as part of some application (e.g. Microsoft Repository works this way with UML knowledge). Other translation modules read knowledge from external source representations (e.g. data bases, flat ASCII files). On the other side of the equation, translator modules may distribute knowledge directly to target applications or to some defined external representation.

In this section, we present examples of "source interface to external target" and "external source to external target" translators. In fact as we shall see, Figure 5.10 represents somewhat of a simplification as the actual translation process sometimes requires multiple mechanisms for acquisition and distribution.

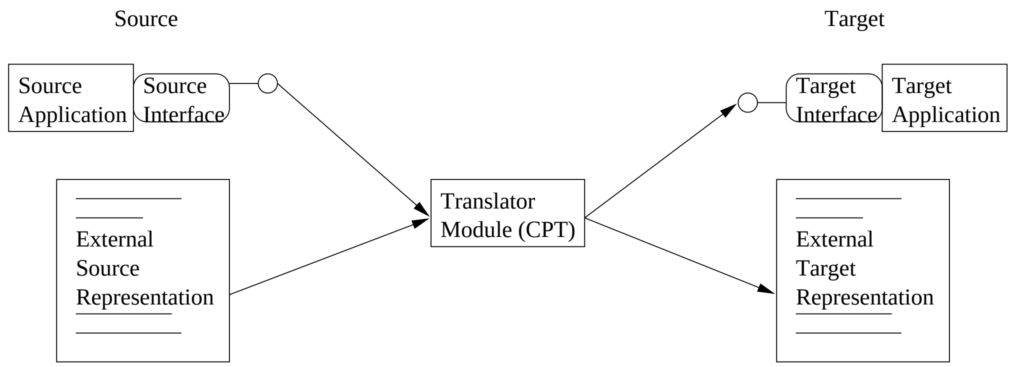

Figure 5.10: Translation module mechanisms for acquisition and distribution

5.5.1 CPM to CPD

The first translation module we will present involves the exchange of knowledge from the requirements analysis phase to the detailed development phase. We will refer to this as the CPM->CPD translator. Effectively this involves mapping the largely graphical requirements from the CPM toolset to the CPL in order to produce an initial domain specification (see Figure 3.1). Recall from Section 3.2.3.6 that the driving subset of the CPM requirements expression for this purpose is the Combined Thread Diagrams (CTD). Typically a combined thread diagram corresponds to a planning domain operator or schema. As we shall see, this isn't always true as a CTD may actually split into multiple operators.

Our implementation of the CPM->CPD translator is built into the CPM toolset. The translator is written in CLIPS [Giarratano, 1994] and uses the HARDY hypertext diagramming meta case-tool interface (or API) to walk through a currently loaded set of domain requirements and to extract the required information. In Figure 5.11 we present an example of a Combined Thread Diagram taken from the three pigs house building domain.

Figure 5.11 illustrates the CTD requirements for the "build house model" domain process. This CTD uses the mutual exclusion notation (i.e. bounding box labelled with a "0") to indicate that there are actually two distinct ways of configuring this process. During the translation process, the CPM->CPD translator will iterate through each CTD and translate it to a cpo-process in CPL. This step also checks to see if the CTD

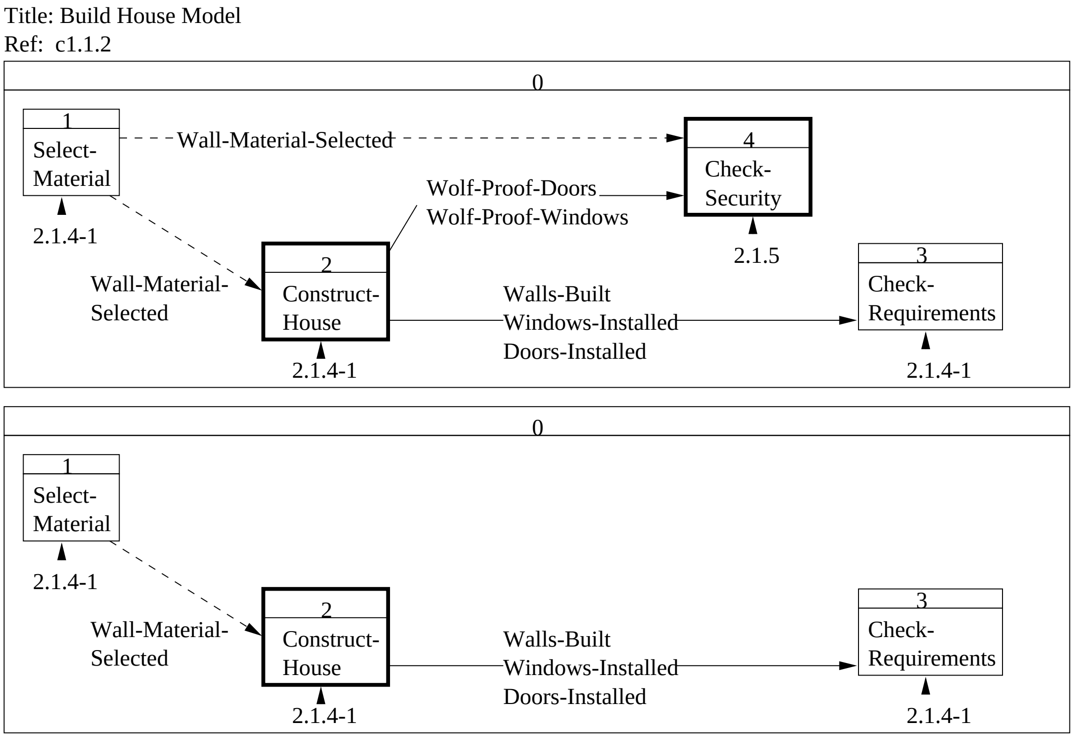

Figure 5.11: Example of a Combined Thread Diagram

will map into multiple cpo-processes due to the use of a mutual exclusion notation.

CPM	CPD
<filename> for translated output	%define-domain(<filename>)
CTD w/o mutual exclusion	cpo-process
CTD with mutual exclusion	cpo-process ... cpo-process
–	cpo-activity-specification
Action node	cpo-action
Control data Data containing information Event data	cpo-input-constraint cpo-output-constraint cpo-ordering-constraint
–	cpo-begin
–	cpo-end

Table 5.1: Basic mappings for CPM->CPD translator

We have outlined the basic mappings from CPM to CPD in Table 5.1. We can see that we first create the %define-domain CPL command based on the target filename. For each CTD, we create a cpo-process sort instance as we have just discussed. Part of this generation also involves synthesising the new activity specification instances which will contain the detailed process constraints. As we can see at the bottom of the table, we also create new cpo-begin and cpo-end instances to give the process an abstract temporal scope. Each "action node" on the CTD diagram is translated to a cpo-action instance. This translation involves synthesising a new label, pattern, and timepoint paring for the action begin/end. In a CTD we identify data flows between action nodes. These flows may indicate control, event or data containing information. In our current implementation we do not attempt to infer what this detailed data flow typing could mean for synthesising complex CPL relationships. Instead, we treat all types by mapping this to cpo-output-constraints for the source, cpo-input-constraints for the target, and we generate a cpo-ordering-constraint between source and target cpo-actions.

This rather coarse but helpful mapping produces the initial domain specification. It is the responsibility of the detailed domain development phase to refine this knowledge into a representation which is more closely aligned with the operational language of an AI planner. For example, while "Select-Material" from Figure 5.11 came across as a mapped cpo-action from the modelled CTD action node, we actually will eliminate this cpo-action in the CDE. The reason for this is that the material selection will actually be handled or represented by a variable binding which will occur during planning. Likewise, we may transform some of these raw input and output constraints which came from the data flow knowledge into more specialised constructs such as cpo-resource-constraints. Thus, the implementation of this translator exhibits the CPF integration between these process domain management steps.

5.5.2 CPD to TF

In Section 5.5.1 we discussed the translation of a subset of results from the requirements methodology to the initial domain specification (expressed in CPL). This bridged the integration step between the requirements analysis phase and detailed domain development. As the architecture in Figure 3.2 indicates, the next integration step addresses the externalisation of the domain knowledge for phases such as process synthesis using an AI planner. In this section we will present our implementation of this translation between the refined detailed domain definition (CPD) and the Task Formalism (version 2.3) [Tate *et al.*, 1994a] which is used in the O-Plan planner.

This CPD->TF translator is basically part of the CPD parsing engine and is an "external source to external target" translation module with respect to the model in Figure 5.10. This contrasts with the CPM->CPD translator which is a "source interface to external target" module. Looking back to our presentation of the CPD parsing engine in Figure 5.3 though, we can see that the user may actually perceive this translation to be either of these two cited types. The reason for this is that the CDE application can directly interact with the CPD->TF translation module inside of the CPD parsing engine simulating an external to external translation which is the same one the user could also initiate from the command line.

CPD	TF v.2.3
cpo-process	action schema
cpo-plan	task schema
cpo-action.pattern()	action node
subsort of cpo-other-node	dummy node
cpo-ordering-constraint	ordering
cpo-output-constraint	effect
cpo-input-constraint	condition
cpo-resource-unit@tfpsv	resource unit
cpo-activity-relatable-object.resource-type()@tfpsv	resource type
subsort of cpo-activity-relatable-object	object type
instance of cpo-activity-relatable-object	resource instance
cpo-resource-constraint cpo-agent.performs-act cpo-agent.performs-proc	resource expression
cpo-always-constraint	always statement
cpo-variable-constraint	schema var statement
cpo-process.expands()	expands pattern
cpo-annotation-constraint	comment
<round trip data>	enhanced comment

Table 5.2: Basic mappings for CPD->TF translator

The translation from CPD to TF basically follows the mapping presented in Table 5.2. In presenting the CPO (see Section 3.3.1), which underpins the CPD file expression, we identified a difference between processes and plans. We can see in fact that in a CPD specification a cpo-process maps to an action or normal schema in TF whereas a cpo-plan indicates a task schema. In translating cpo-actions which are included in a cpo-process or cpo-plan's activity specification we can see that the TF side utilises the cpo-action pattern. Dummy nodes are created from the subsort cpo-other-nodes which are also included in the activity specification. The cpo-ordering-constraints are syntactically mapped into TF schema node orderings. Cpo-input-constraints and cpo-output-constraints are pretty much directly mapped which means that their expressions in CPL very closely resemble the legal grammar for their counterparts in TF. Note though that substitution of node references is performed as needed because TF uses node numbers to refer to schema nodes and these node numbers did not exist prior to translation. For example, consider the following before and after expressions.

- before (CPD): constraint.expression(Obj26inp0)="supervised {walls built} at Obj26n3 from [Obj26n2]")
- after (TF): conditions supervised {walls built} at 3 from [1];

Obj26n3 and Obj26n2 is nonsense in the newly generated TF file. A pass through the expression locates references and replaces them with the new node numbers assigned during the node list building. The next two mappings rely on the TF extension to the core CPO (see Section 3.3.2.19). We denote this by using an "@tfpsv" package identifier. In particular we map cpo-resource-units to domain-level resource units in TF (e.g. a resource unit for the resource "money" may be "pounds"). The extension function "resource-type" is used to associate an ARO with a TF type such as "consumable_strictly". TF object types are mapped in as new subsorts ("isa") of the cpo-activity-relatable-object sort. Instances of these subsorts become the resource instances in the TF domain file. Resource expressions declared in TF schemas can actually come from either cpo-resource-constraints included in the activity specification or from the performable relations associated with cpo-agents. TF always statements, schema variable statements, and TF comments come over from cpo-always-constraints, cpo-variable-constraints, and cpo-annotation-constraints, respectively. The expanding TF pattern for a schema is retrieved from the "expands" cpo-process function.

The final entry in Table 5.2 represents work on one aspect of what has been called the "round-trip" problem in knowledge sharing (see [Chan, 1995]). In particular we are interested in providing a way to preserve or embed knowledge of the items that cannot be translated directly to O-Plan TF (or any target language). The idea is that if we would need to create the opposite direction translation module (i.e. TF to CPD) we could write it in a way that will recognise and restore these untranslatable elements. Note though that in embedding this knowledge we cannot violate the existing grammar of TF or else it won't be parsable by the default TF parser.

Thus the idea is that we can embed "enhanced" comments. Let's say we have some instance of a process domain model, M0, expressed in CPL. Let's also say we have two translators (T1,T2), one which translates between CPL<->TF (M1) and one for another operational domain language such as CPL<->Act (M2). We will say there is some construct, e.g. an instance of a custom constraint type, let's call it Q1, which is currently expressed in M0. It is then the responsibility of T1 and T2 to encode Q1 in a translation to M1 and M2 respectively in such a way that Q1 is parsable but also retrievable. For example, T1 might translate Q1 into

;;! preference-constraint (Q1) = 'prefer resource bricks'

This would appear as an ordinary comment to the TF parser, but the T1 parser would recognise the ";;!" as being different than the ";" construct and return this to

SORT preference-constraint={Q1}
constraint.expression(Q1)="prefer {resource bricks}"

in M0′ (i.e. the round-trip copy of M0). T2 likewise could use the Act formalism [Wilkins and Myers, 1995] comment syntax to form such an enhanced comment. Thus this translator integrates the domain development and process synthesis phases in CPF.

5.5.3 OPO to CPL

The final translation module, OPO->CPL, which we present in this section is another "external source to external target" implementation. In this case though we are providing an example translation in the opposite direction of what we presented in Section 5.5.2. Specifically we are interested in bridging from a process synthesis phase to a process management phase by mapping the results from an AI planner (i.e. an external framework tool) into CPL. This translated knowledge will be used in the Common Process Editor (see Section 5.3). This translation module is part of the decoupled OPO (O-Plan Plan Output Format) parsing engine (see Figure 5.6). As with the CPD->TF translator, this translation process may be run from within the CPE or as an external command line program.

The OPO format is a rather sparse approach to exporting knowledge of a generated plan. It is defined in a subsection of the Task Formalism user guide [Tate *et al.*, 1994a].

OPO basically consists of a list of nodes in the generated plan along with information on their ordering. In order to make more of the knowledge of the plan available we created a new OPO export module for O-Plan which can be loaded at runtime. This new module can also export a plan's GOST, TOME, (see Section 2.3.1.2) and resource entries.

OPO ext.	CPL
plan ... end_plan	cpo-plan
–	cpo-process
node ... end_node	cpo-start cpo-finish cpo-action
–	cpo-begin cpo-end
predecessor and successor lists	cpo-ordering-constraints
always TOME entry	cpo-always-constraint
node TOME entry	cpo-output-constraint
GOST entry	cpo-input-constraint cpo-output-constraint
resource_entry ... end_resource_entry	cpo-resource-constraint

Table 5.3: Basic mappings for OPO->CPL translator

As we have done in Sections 5.5.1 and 5.5.2, we present a table of the basic mappings for this translation in Table 5.3. All of the constructs in an OPO file are bounded by the plan/end_plan pairing which corresponds to the single cpo-plan instance created for the CPL translation. Enclosed within this pairing are a series of node/end_node pairings which corresponds to either cpo-start, cpo-finish, or cpo-action sort instances. The OPO node_type indicator is used to differentiate. Predecessor and successor lists in the node/end_node scope are used to synthesise cpo-ordering-constraints.

One of the interesting issues involved in this translation is the mapping from one single, "flattened" OPO plan to the CPE-style "one (sub-)process per window" approach. The various instances of subprocesses (cpo-process) or cpo-node expansions in the OPO file needs to be detected and reconstructed for the CPL representation. Using the OPO format for naming node references (e.g. node-3-1, node-3-1-1) we can detect which nodes belong in the activity specification of an expanded process. In addition to this, cpo-begin and cpo-end nodes need to be synthesised and added as well, bounding subprocess intervals.

As we mentioned earlier in this section, we have enriched the OPO format to also include knowledge of the conditions, effects, and resource utilisation. This knowledge is translated into the appropriate CPO counterparts as illustrated in Table 5.3. Thus, the implementation of this translator demonstrates the CPF integration between process synthesis and process management. Similar translators can be built to map the results of other process lifecycle tools (e.g. process evaluation tools) into CPL.

5.6 Conclusion

> *At this point we have presented the methodology and design of the CPF architecture and our implementation of the components which were introduced in Section 1.2.3. We briefly summarise this fact and point toward the analysis of the framework in Chapter 6 which uses the thesis process scenarios.*

In this chapter we have discussed the our implementation of the design of the Common Process Framework. Throughout this discussion we have shown how this work utilises, combines and extends research which we cited and presented in the literature review. Chapter 3 provided us with a general overview of the phases and generic architecture of the CPF in Section 3.1. This chapter can then be viewed as a detailed implementation of this architecture.

Back in Chapter 3 we discussed our adaptation of CORE to act as the requirements methodology. We then examined the concrete representational choices as we laid out the process ontology and process language. We also presented our notion of a complete solution to motivate the need to interleave process rationale knowledge along with the process design artifact. Finally, we provided a review of each of the tools implemented within the CPF toolset.

Our next step is to demonstrate the application of the framework described in this chapter for a portfolio of scenarios which represent realistic applications of this approach. These applications span a variety of organisations and show how we can use this approach to manage rich organisational process knowledge.

Chapter 6

Analysis of Scenarios and Requirements

This chapter presents a portfolio of scenarios across a range of domains which illustrate the intended uses of the ideas developed in this thesis. The domain of each scenario is described and the underlying concepts are outlined. We then elaborate on how we utilised the components of the integration framework and our research methodology, which we described over the previous chapters, to manage the process knowledge for each scenario. Finally, we consider our approach in light of the requirements we compiled for our framework.

6.1 Introduction

This section discusses our rationale for compiling a portfolio of scenarios and a set of requirements for the purpose of motivating, guiding, and evaluating our approach.

The process of conducting applied research typically begins with a vague idea of what the research question is and what the proposed avenue of approach will yield for particular applications. As we outlined in Section 1.2, our driving research question is "How can we improve the methodology of synthesising and managing organisational process knowledge?". A central thrust of our proposed avenue of approach sought to draw on the tools, techniques, and representations developed for AI planning.

As Cohen points out in his handbook on empirical methods for Artificial Intelligence [Cohen, 1995], researchers tend to use "exploratory studies" which help to develop unrefined ideas and vague questions. Cohen characterises these studies as being similar to working in test kitchens. Interaction with these studies influences the goals of the research and helps to form the nature of the solution. This process of research interaction with exploratory studies has a parallel in software engineering called scenario analysis [Hsia *et al.*, 1994, Kazman *et al.*, 1996].

6.1.1 Scenario Analysis

During the various stages of design, development, deployment, and maintenance of software systems it can be very helpful to engage in scenario analysis. One of the most well-known and widely applied scenario analysis approaches is the use case analysis method developed by Jacobson [Jacobson *et al.*, 1992]. Scenario analysis has been defined in a software engineering context as

> "the process of understanding, analysing, and describing system behaviour in terms of particular ways the system is expected to be used." [Hsia *et al.*, 1994]

In the case of AI research-based exploratory studies we can amend this definition of scenario analysis to read:

> "the process of understanding, analysing, and describing knowledge representation, tools, methods, and techniques in terms of the particular ways the research approach is expected to be applied."

So, as in software engineering, we can utilise a portfolio of scenarios to explore the applications of our ideas to various domains. The selection of the particular scenarios helps to define the scope of the work. In addition, these scenarios partially help to validate the approach.

Scenario-based validation of the approach is partially justified by the fact that the scenarios we worked with were not simple toy situations developed for our own research purposes. In fact, four of the scenarios which we describe in this chapter were written as part of our involvement with other research projects during the period of thesis study. These scenarios were identified as good candidates for both the needs of this thesis research as well the requirements of the participating project. These projects address processes involved in business, manufacturing, and military operations. The applied project goals thus served to provide realistic requirements on the scenarios which were developed.

6.1.2 Requirements

In addition to the portfolio of scenarios, we compiled a wide-ranging set of requirements for a framework which encompasses the synthesis and management of organisational process knowledge. The sources for these requirements ranged from some of the specific projects, applications and research papers which we reviewed in Section 2.3.4.2 to more general inputs derived from workshop and conference discussions and interactions with organisational process modellers. These requirements are categorised and discussed in Section 6.3.

6.1.3 Summary

We chose to structure our research around a portfolio of scenarios which we developed with an intent to use in both this thesis research and in our involvement with other projects. In Section 6.2 we provide a high-level description of the scenarios which are then detailed in Sections 6.2.1 through 6.2.5. For each scenario, we examine the representational concepts which were involved and then report on the application of our framework. Section 6.3 presents the set of requirements we compiled and discusses how some of the requirements are met by using the Common Process Framework.

6.2 Scenarios

We introduced the scenario-based approach above. This section presents the scenarios which we will use to illustrate applications of the Common Process Framework along with a brief background on their development. Following the high-level descriptions in this section, we will explore each scenario in turn.

- **Three Pigs Building** (Section 6.2.1): The three pigs **building scenario** is the most basic example we have used to validate and test our approach. It involves the synthesis and representation of a process for building a house in a domain where cost and material constraints must be evaluated. It is based on the 3-pigs AI planning domain which was built for a simple demonstration of O-Plan in 1993.

- **Supply Chain Reengineering** (Section 6.2.2): The supply chain reengineering engagement is a **business scenario** which we developed [Polyak, 1998e] that involves a set of business consultants that wish to model and simulate inter-company supply processes. This scenario was developed to focus the work of the Process Interchange Format (PIF) during 1998 (cf. [Polyak *et al.*, 1998]) in addition to its use here in this thesis. The content of the scenario was based on a supply chain demonstration presented by the Workflow Management Coalition (WfMC) at the 1996 Business Process and Workflow Conference.

- **Microwave T/R Process Plan**[1] (Section 6.2.3): We adapted a microwave transmit/relay **manufacturing scenario** [Polyak, 1998d] in order to illustrate the representation of a realistic, shared process plan. A microwave transmit/receive (T/R) module is an electrical component that can be found in modern telecommunication devices designed for scientific and commercial long-range defence applications (e.g. radar, satellite communications, long distance television and telephone signal transmissions). This scenario was used in 1998 during the development of the National Institute of Standards and Technology's (NIST) Process Specification Language (PSL). Our sources for this manufacturing process plan knowledge included the EDAPS process planning module [Smith *et al.*, 1996, Smith, 1997] and Sander's text on the T/R module product specification [Sander, 1987].

- **Camile Manufacturing Interoperability**[2] (Section 6.2.4): In addition to the microwave T/R process plan representational scenario, we de-

[1] This work was conducted under a grant from the U.S. Department of Commerce entitled "Scenario Development for Shared Process Models", award number 70NANB8H0009. Principal Investigators: Steve Polyak and Austin Tate, Artificial Intelligence Applications Institute (AIAI).

[2] Also part of the work on the U.S. Department of Commerce award number 70NANB8H0009.

veloped an interoperability **manufacturing scenario** for NIST's PSL project [Polyak and Aitken, 1998]. This scenario outlined the exchange of process knowledge in a manufacturing environment between a process plan modeller and a job shop scheduler. The manufacturer is a fictitious model car company, the Camile Motor Works. The product, processes and factory knowledge presented in this scenario were based on a planning and control reference case study developed by members of the Waterloo Management of Integrated Manufacturing Systems (WATMIMS) Research Group in 1991 entitled "Intelligent Manufacturing Management Program State of the Art Scheduling Survey" [McKay and Moore, 1991].

- **Military Processes** (Section 6.2.5): We investigated the potential application of part of the CPF approach to a U.S. Army **military scenario** that was being developed as part of a small unit operations (SUO) O-Plan transition project. The interesting aspect here has to do with the fact that the U.S. Army already has a well-defined Military Decision-Making Process (MDMP). In addition, we briefly mention the role of CPF tools in the ACP3 Air Campaign Planning Process Panel approach for the DARPA/Air Force Research Laboratory (Rome) Planning Initiative (ARPI) TIE 97-1 in which an early version of CPE was used alongside a COA evaluation matrix.

6.2.1 Three Pigs Building

This section describes the three pigs building scenario which we use to illustrate an application of our thesis approach which structures the synthesis and management of house-building process knowledge. We begin with a description of the domain followed by a review of the modelling concepts required. We then present a review of the products produced for this scenario.

One of the standard Task Formalism demonstrations developed for O-Plan in 1993 involved a house-building domain based on the familiar "three pigs" children's tale. In this tale, each pig builds a house from one of three possible building materials: straw, sticks, and bricks. Of the three, only the brick material will provide adequate support to protect a pig from a wolf. This basic idea was adapted and extended for the AI planner to provide examples of a building domain which involved the representation and interaction between resources used for cost and materials. Various task configurations were developed to illustrate the interplay between the requirements of security and building expense.

The three pigs domain thus centres around the activities involved in building a pig house along with knowledge of the resources required for executing the house building processes. These activities may range from more abstract processes such as "build house" to more detailed actions such as "purchase straw".

We introduced some of the details on this scenario back in Section 1.2.1. We reproduce the basic building options, materials and costs for our variation of the three pigs domain in Figure 6.1. We also revisit the list of domain requirements summarised in Table 6.1.

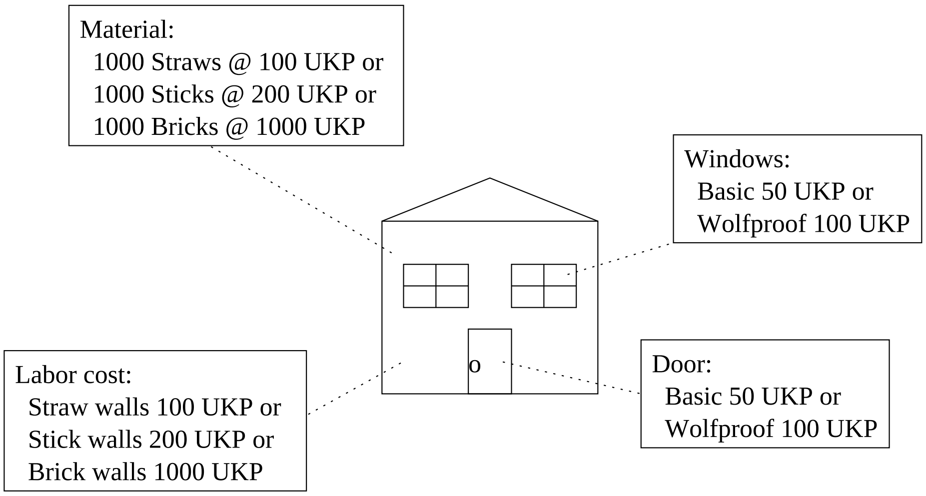

Figure 6.1: Pig house options

In general, we are only interested in one high-level task for this domain. Note that our use of the word "task" here refers to the equivalent of a "task schema" in O-Plan. This task is to simply build the required house subject to any constraints which are included in the task definition. For example, the task may be to build a house for less than or equal to 500 UKP.

R1.	*A house requires windows, walls, and a door.*
R2.	*Walls must be built from 1000 units of straw, sticks, or bricks.*
R3.	*Wall material must be homogeneous.*
R4.	*Bricks walls are wolfproof.*
R5.	*Windows may either be basic or wolfproof.*
R6.	*Doors may either be basic or wolfproof.*
R7.	*A secure house must have a wolfproof door, wolfproof windows and the walls must be made from wolfproof material.*
R8.	*1 brick costs 1 UKP.*
R9.	*1 stick costs 20p.*
R10.	*1 straw costs 10p.*
R11.	*Brick walls incur 1000 UKP for labor.*
R12.	*Stick walls incur 200 UKP for labor.*
R13.	*Straw walls incur 100 UKP for labor.*
R14.	*Labor and parts for a basic door is 50 UKP.*
R15.	*Labor and parts for a wolfproof door is 100 UKP.*
R16.	*Labor and parts for a basic window set is 50 UKP.*
R17.	*Labor and parts for a wolfproof window set is 100 UKP.*
R18.	*Wall material must be purchased before walls are constructed.*
R19.	*Walls must be constructed before windows or doors are installed.*

Table 6.1: Pig House Domain Requirements

Given these domain requirements, we should be able to construct an AI planning domain which adheres to these constraints and which will enable the automatic or semi-automatic synthesis of a customised house building process. So our scenario is simply to conceptualise and define the initial house building domain requirements, operationalise the domain for use in an AI planner, synthesise a house building process and to visualise the set of actions. In the following section, we explore the underlying representational concepts that these requirements presuppose.

6.2.1.1 Concepts

This three pigs building domain requires the representation of a basic set of process and process-related concepts. These concepts are outlined in the following list. For each concept, we cite an example and also refer to a domain requirement which either directly or indirectly implies the example of the concept.

- **actions/processes**: An example is an action representing the building of walls. This is implied by requirement R2.
- **conditions**: A condition might be "walls-built". This could be applied to actions for installing windows and actions for installing doors in order to satisfy requirement R19.
- **cost**: Cost is actually a special case of resource requirements, but we can see examples of this concept in requirements R8-R17.
- **decomposition/expansion**: Decomposition isn't strictly called for in the domain requirements. Its use though could help to structure the varying levels of detail which we find in the requirements listing. For example, R1 suggests some high-level "build-required-house" task or process which can then be decomposed into more detailed steps.
- **dependency**: Some actions depend on others, such as the link between wall building and installation of doors and windows. This is implied by R19.
- **effects**: The process of building walls should assert some effect such as "walls-built" in order to satisfy the condition for installing windows and doors (e.g. R19).
- **evaluations**: The evaluation of variables and world states are implied in R7.
- **preferences**: Preferences are only indirectly implied by the various choice points posed by the domain requirements. For example, R2 states a choice of building material and therefore a possible preference for selection.
- **resource classes**: There appears to be two classes of resources, one involving capital (e.g. R8) and another which involves building material (e.g. R2).

- **resource instances**: There are instances of resource classes. For example, a quantity of straw, sticks and bricks are all instances of building material for this domain (e.g. R2).

- **resource requirements**: The process of building brick walls requires 1000 UKP. This is stated in R11.

- **resource units**: The money resource appears to have two possible units either pence (p) or pounds (UKP). This is given by R8-R17.

- **resource types**: An example resource type is "strictly consumable". This is partially implied for a resource class like money by requirements R8-R17. This is reinforced by the fact that there are no domain requirements for actions which produce money.

- **tasks**: The domain task of building a required house is implied by requirement R1.

- **temporal relationships**: The activity of purchasing of wall material must occur before the activity of constructing the walls. This is given in requirement R18.

- **variables**: An example is a variable which will be needed to record the selected building material. R3 implies there to be one such variable which can only be assigned one value.

- **world states**: A world state in which a constructed house may be secure can be evaluated by checking fluents which specify information about the door, windows, and building material. This is implied in R7.

6.2.1.2 Applying CPF

Given the description of this domain and a high-level understanding of the requirements and concepts for this scenario we can consider an approach towards managing this rich house building process knowledge. This approach should integrate the information amongst various phases and tools involved in this effort. Looking back to Section 6.2.1, we can see that we have outlined a rough idea of what these main steps should involve

- conceptualise and define the initial house building domain requirements
- operationalise the domain for use in an AI planner
- synthesise a house building process
- visualise the set of building actions

These steps map directly onto the CPF phases presented in Section 3.1.1. They correspond to **requirements analysis**, **detailed domain development**, **process synthesis**, and **process management**.

In the **requirements analysis** phase, we applied the Common Process Methodology (see Section 3.2) to this domain in order to elicit and structure our knowledge of it. This guided us from our more abstract understanding of the domain down towards the richer content of the house building combined thread diagrams (see Section 3.2.3.7). From this catalogue of combined threads we can index back up the hierarchy of detailed knowledge (CTP->ITD->TED->VSD) to understand the context and role of each individual house building operation.

As we entered our next phase of operationalising the domain knowledge (i.e. **detailed domain development**) we drew on our base building block representation which we developed in CPM. The integration of these phases was supported by the translation from CPM to CPD (see Section 5.5.1). Using the Common Domain Editor (see Section 5.2) we further refined this initial building domain specification and made operational decisions such as how to implement process variables and patterns and how to express detailed constraints. We utilised extensions to the core language (see Section 3.3.2.5) to capture information about these decisions (see Chapter 4) and to support specialised tool-related aspects. These tool-related extensions were supported by a specialised expression-builder software module (see Section 5.2.3, plugged-in at runtime to the domain editor).

Again, CPF assisted in the integration with our next phase, **process synthesis**. Using the CPD to TF translator (see Section 5.5.2) we generated an appropriate domain input for the O-Plan planner. Using O-Plan, we loaded this knowledge, selected the main house building task and developed a course of action for executing house construction. This execution was constrained to meet the cost and security requirements defined in the top level process. We loaded and executed the extended O-Plan output (OPO) module which we designed to provide richer plan export information from O-Plan.

Finally, the newly synthesised construction process entered the **process management** phase through the OPO->CPL translation. As we discussed in Section 5.3 this phase is supported by the Common Process Editor. The house building process may be viewed level-by-level, inspecting each of the detailed constraints and dependencies attached. If the process synthesis phase (either mixed-initiative or automated) had been able to produce knowledge of its design rationale (e.g. as we have shown in Chapter 4) the house-building organisation would be able to inspect this as well.

As we have seen, the CPF has been applied to the problem of effectively managing house building process knowledge. While the final house building domain only consisted of 18 simplified schemas (or operators, acts, etc.) and the newly synthesised process only consisted of 1 plan, 4 subprocesses, and 18 nodes this served as a reasonable template scenario for demonstrating the CPF approach. In the following scenarios we re-apply this approach to more realistic domains.

6.2.2 Supply Chain Reenginering

This section describes the Supply Chain Reenginering engagement which we use to illustrate an application of our thesis approach which structures the synthesis and management of supply chain process knowledge. We begin with a description of the domain followed by a review of the modelling concepts required. We then present a review of the products produced for this scenario.

The goal of the Process Interchange Format (PIF) Project is to develop an interchange format to help automatically exchange process descriptions among a wide variety of business process modelling and support systems such as workflow software, flow charting tools, process simulation systems, and process repositories. As an example of such an exchange, we created a demonstration scenario during this thesis development period which described the use of PIF in the modelling and simulation of an integrated supply chain where different companies co-operate through a global supply chain management procedure (cf. [Gattorna and Walters, 1996, Arntzen *et al.*, 1995, Lee and Billington, 1993]) to deliver commercial electronic goods. This scenario described the possible exchange of process knowledge between a business process modelling tool/library[3] and a process simulation package[4] with PIF acting as the interlingua.

This scenario was adapted from the Workflow Management Coalition's (WfMC) workflow interoperability demonstration presented at the 1996 Business Process and Workflow Conference in Amsterdam. This scenario work provided the PIF group with a framework for evaluating, challenging and extending the elements defined within the PIF-Core.

6.2.2.1 Supply Chains

A supply chain is essentially a network of facilities and distribution options that performs the functions of procurement of materials, transformation of these materials into intermediate and finished products, and the distribution of these finished products to customers [Lee and Billington, 1992]. There are supply chains in both service and manufacturing organisations. The complexity of the chain may vary greatly from industry to industry and company to company. Traditionally marketing, distribution, planning, manufacturing, and purchasing organisations along the supply chain operated independently. This independence typically meant that there wasn't a single, integrated plan for the organisation. There were as many plans as businesses. A need existed for a mechanism which integrated these different functions. Supply chain management is now referred to as the strategy through which this integration can be achieved. This has become an important issue for many organisations as they rethink the way they do business. For example, Hammer and Champy pointed out a need for radically changing the processes of a manufacturing logistics supply chain in their pioneering book on Business Process Reengineering [Hammer and Champy, 1994].

The overall objective of the supply chain for this scenario is: to obtain benefits by rapidly getting manufactured commercial electronic products from the production line into retail stores. In order to ensure that this objective is met in an effective way, these processes may need to be modelled, synthesised and simulated across organisational

[3] Massachusetts Institute of Technology's (MIT) Process Handbook
[4] Knowledge Based System Inc.'s (KBSI) ProSim

boundaries. This process may be facilitated by providing a way to automatically or semi-automatically exchange process descriptions between a modelling tool and simulation tool using CPF.

6.2.2.2 Domain Objects Overview

In the main scenario document [Polyak, 1998f], we discussed the responsibilities of the companies which were involved in the modelled supply chain. We then widened the scope to present the additional elements which were needed to represent these processes. These objects were modelled in a simple UML [Booch *et al.*, 1998] object model to highlight a taxonomy of entities and relations between them. A high-level model showing some of the supply chain scenario objects is depicted in Figure 6.2.

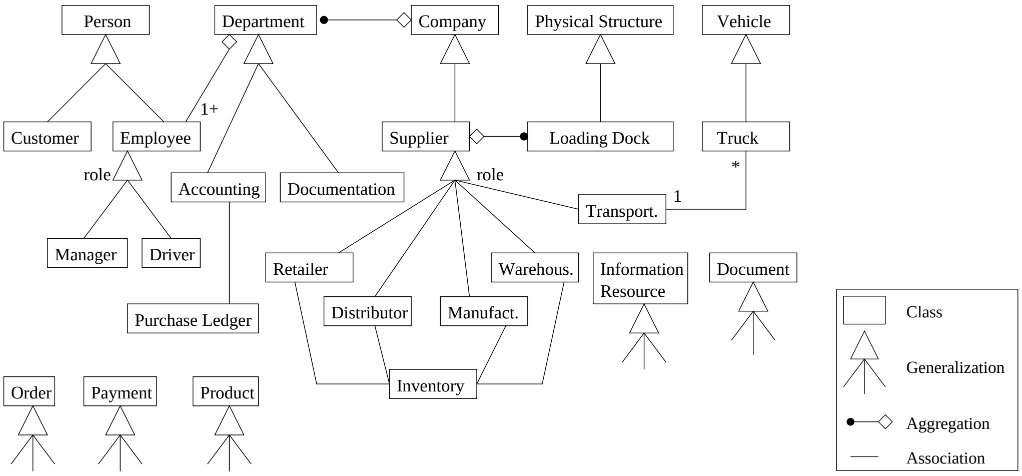

Figure 6.2: Partial UML object model of the supply chain entities

Starting toward the top of the model, we can pick out two fundamental classes of entities: **company** and **person**. Supply chains are essentially centred around these basic concepts. People involved in these simplified processes may be **customers** or **employees**. Looking at companies, we can see that a company is typically composed of zero or more **departments**. These departments contain one or more employees which carry out the specific tasks. Employee types are usually associated with the nature of the task which they perform (e.g. a **driver** transports **products**, a **manager** manages other employees, etc.) Departments may require specific objects to carry out their tasks. For example, the **accounting department** requires a specialised record, the

purchase ledger, for maintaining the company financial records.

Companies involved in supply chain management are referred to as **suppliers**. In this domain, there are 5 supplier types or roles as described above. Suppliers require objects such as **trucks**, **loading docks**, and records of current stock (i.e. **inventory records**). Suppliers communicate and perform various transactions by using a variety of **document** types. These physical documents are linked to **information resources** which are, in turn, related to various abstract business objects, such as **orders**, **payments**, etc. The following sections address some of the objects presented in Figure 6.2 in more detail.

6.2.2.3 Process Overview

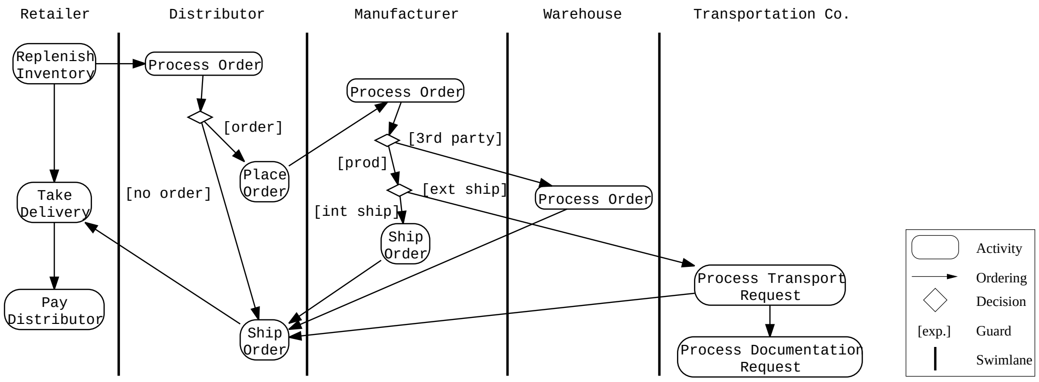

Figure 6.3: High-level UML activity model of the supply chain processes

A high-level model of the required cross-organisational supply chain process is shown the UML activity diagram in Figure 6.3. This diagram has a "swim lane" layout which identifies the temporal ordering of the processes across all of the companies. Each process identified in this diagram is associated with a particular supplier and is broken down in the source scenario document [Polyak, 1998e].

The flow of supply chain activities stem from a "replenish inventory" process which is initiated by the retailer. This leads to a cross-organisational activation of a process at the distributor. A key decision taken by the distributor at this point has been highlighted. This decision involves either satisfying the order via existing stock or by requesting products from the manufacturer. While the former simply requires a shipment to the retailer, the latter involves placing an order with the manufacturer. The manufacturer, in turn, makes a couple of important decisions while processing an order. These decisions are to either request stock to be sent from a third party warehouse or to satisfy the order via a scheduled production run. For orders completed at the factory, a decision is made as to how the product will be shipped to the distributor. The manufacturer typically requests pickup and delivery from a transportation company, but it also has a limited capacity to deliver products on its own (usually only performed for smaller orders). The transportation company handles the documentation for product shipment along with providing the transport service. Once the distributor receives the products, they are sent along to the retailer. The retailer completes the modelled process by sending payment for the goods to the distributor.

6.2.2.4 Applying CPF

While this business process scenario appears to be very different from the building process scenario we discussed in Section 6.2.1 we found that a similar application of the Common Process Framework was very effective and useful for this situation as well. The basic approach and execution of phases (i.e. **requirements analysis**, **detailed domain development**, **process synthesis**, and **process management**) served to improve the overall methodology of eliciting, constructing and managing various configurations of the overall supply chain process.

As in the building domain, we began with the **requirements analysis** phase utilising the CPM. After executing the defined methodological steps and identifying the various individual threads in the domain, we united our understanding from the separate viewpoints into a set of 24 unique combined thread diagrams. During this process, moving through each phase in CPM, the methods helped us flesh out the interrelationships of the activities in the supply chain. This facilitation was largely due to our ability to delay in committing to detail about how exactly the configuration would look until we had a general idea at each level (i.e. VBD, VSD, TED, ITD, CTD). For example, it was only at the ITD level that we considered some of the control branching that was needed. The various levels produced can be used as domain browsing tools for domain experts depending on what amount of detail is required. For example, TED-level browsing can give you a good high-level perspective with details in the ITD and CTD.

In this scenario, we found it helpful to use an "environment" bounding viewpoint in CPM which we used to model assertions of global information for the domain. For example, acquiring or asserting lists of suppliers, DRP, MRP, or inventory information. In many cases, actions performing these global assertions connected to the "environment" viewpoint were eventually transformed into either input or output constraints (conditions/effects) (put/get-purchase-ledger-info), always constraints ("{distrib-stock-level}=inadequate"), or to the presence of various ARO instances (TransportCompany1, TransportCompany2, etc.).

During our work with CPM we noted that our modelling was being influenced by our knowledge of the planning system's capabilities (see Section 2.3.7.1). For example, one of the major influences was our anticipation of HTN-style expansion in O-Plan. The mutual exclusion constructs we created in the combined-thread diagrams[5] and overall viewpoint structure diagram defined such decompositional structure. Additionally, we noted parts of the supply chain domain which would best be addressed by embedding contingency-based actions (cf. [Draper *et al.*, 1994]) into a generated supply chain network of actions. As we anticipated the lack of such constructs using O-Plan, we chose to move such bifurcating decisions up into the domain design time rather than trying to express this process runtime evaluation.

As in the building scenario, we automatically translated the requirements into an initial specification and during **detailed domain development** (which utilizes the CDE) we refined this knowledge by continuing to make more detailed design decisions. We defined the concrete activity relatable objects (and their corresponding process variables) which were referenced throughout the various input/output requirements relationships. Ontological extensions (e.g. order is an activity-relatable-object, distributor is an agent) were defined for this scenario to specialise the representation of these entities (i.e. those listed in Figure 6.3). These were all packaged in a single "supply chain" CPL extension with only minimal identifying definitions. We also identified the global state information (e.g. distrib-stock-level) which would be used to synthesise different supply chain process configurations depending on their particular values. These various configurations could then be shared with tools which would simulate/animate

[5] The CPM mutual exclusion constructs in CTDs are very similar to the Process Handbook alternate process decompositions [Malone *et al.*, 1993]

the resultant process.

During **process synthesis** we practised generating various domain configurations (e.g. no transportation companies available, distributor inventories adequate, MRP not feasible) based on different domain criteria. This was meant to simulate a team of business consultants making various changes to the domain and inspecting the result on the newly generated processes. Within each of these tests, the O-Plan planning system was used to also inspect the various feasible plan/process configurations for a given domain configuration. Resultant process configurations were exported for viewing in the CPE.

While in the building scenario we focused on a single pass through the CPF phases, we note that in this business scenario we have introduced a loop between the use of the CDE, an external AI planning system, and the CPE (i.e. **process management** phase). Following each newly synthesised process configuration, we imported this knowledge into CPE for process inspection. We then looped back to use the CDE to generate a new domain configuration. This was translated for use in O-Plan and the output from the planner was then translated again back into CPL. As CPE allows us to manage multiple, open processes we could flip from one process to the other, creating an integrated catalogue of possible supply chain processes.

6.2.3 Microwave T/R Process Plan

This section describes the Microwave T/R Process Plan representation scenario which we use to illustrate an application of our thesis approach which structures the visualisation and management of process knowledge for building an electro-mechanical product. We begin with a description of the domain followed by a review of the modelling concepts required. We then present a review of the products produced for this scenario.

In phase three of the National Institute of Standards and Technology's (NIST) Process Specification Language (PSL) project we developed a representational scenario [Polyak, 1998d] to test the adequacy of the primitive PSL concepts and to assist in pointing out the concepts that may be missing from the available core set. In fact, the process scenario was used in a number of ways to meet the objectives of the NIST project. Some uses of this scenario were to

- Assist in making PSL goals concrete
- Describe mechanisms of extension, representation and translation
- Ensure that PSL is capable of addressing realistic applications
- Serve as a practical example to potential end-users
- Aid in the understanding of the language
- Attract attention to PSL
- Assist in the development of the language's semantics and presentation

This scenario is slightly different than the three pigs domain described in Section 6.2.1 and the supply chain domain in Section 6.2.2. Our focus with this scenario was simply to illustrate the *representation* of the process plan which is provided in Appendix E. In particular, we were interested in its expression in CPL as well as its presentation in CPE. We selected this publicly available description of a realistic process plan domain which is used in manufacturing a microwave transmit/relay module at Northrup Grumman. This domain was initially developed for the EDAPS process planning module [Smith *et al.*, 1996, Smith, 1997]. In addition to this, we utilised Sander's text on the T/R module product specification [Sander, 1987]. While our focus is not on the requirements analysis or detailed domain development phases for this scenario it is still necessary to describe the product and concepts which are involved.

6.2.3.1 Product Overview

A microwave transmit/receive (T/R) module is an electrical component that can be found in modern telecommunication devices designed for scientific and commercial long-range defence applications (e.g. radar, satellite communications, long distance television and telephone signal transmissions). These modules are complex devices having both electrical and mechanical properties.

The following excerpts from [Smith *et al.*, 1996] provides an overview of the various elements which make up a microwave T/R module. For more information on microwave components, see [Sander, 1987].

- The **dielectric** is the substrate on which the artwork is laid out, and on which the hybrid components are assembled. The dielectric serves as a wave-conducting medium. Common materials used are PTFE (Teflon), polyolefin, and aluminium-oxide ceramic.

- The **ground plane** is a metallic layer on top of which the dielectric layer resides. The ground plane is usually made of copper or aluminium. It provides grounding for the circuit and mechanical strength for the device, and it acts as a medium to conduct away heat generated by the device. The heat flux of components in MICs, especially those that transmit power to transmitters, is generally very high on the order of 10-1000 MW/cm^3. Therefore, heat sinking is critical to the performance of the device. The ground plane is the mounting surface for the hybrid components. Thus, machine features such as milled pockets and drilling holes are developed on the ground plane.

- The **artwork** is an etched circuit pattern containing traces, pads to mount hybrid components, components that are directly fabricated on the circuit, fiducials, and reference text elements. Usually, the artwork forms the topmost layer of the dielectric.

- **Transmission lines** are traces that carry energy to different parts of the circuit. ... [There are several possible configurations off transmission lines.] The microstrip configuration is the simplest to manufacture.

- **Vias** are through-holes in the dielectric that connect the upper layer to the bottom of the ground plane. Vias also conduct heat from the upper artwork layers to the heat sink.

- **Surface-mount components** are hybrid elements that are assembled on the surface of the dielectric. The leads of these components do not go into the dielectric (as opposed to the leads of through-hole components, which go to the surface).

- **Mounting features** are usually milled pockets that are used as recesses in which surface-mount components will sit. These pockets are especially necessary for components that dissipate high heat, because these components need to be directly connected to the heat sink.

6.2.3.2 Materials/Tools Overview

Various materials and tools are used throughout the manufacturing of microwave T/R modules at each of the work centres and are referenced in the description of these manufacturing processes. Listed below is a breakdown of some of the items used. These materials and tools may be consumed, reused, worn down, etc. throughout the various steps. These materials are also referenced in the concepts section following the detailed manufacturing process descriptions.

- work centre: Vertical Machining Centre (VMC)
  - board, drill bit, end milling tool, side milling tool, slot milling tool, fixture
- work centre: Electrical Centre (EC)
  - board, pumice stone, scotch-brite pad, oven, photoresist, phototool, spinner, spray, mylar
- work centre: Plating Centre (PC)
  - board, plating resist, copper plating, copper ion solution, tin-lead alloy
- work centre: Manufacturing Centre (MC)
  - board, oven, screen, solder paste, solder flux, component, brush, vapour spray, liquid solvent, alcohol solution
- work centre: Testing Centre (TC)
  - pre-encapsulated board, post-encapsulated board
- work centre: Hermetical Sealing Centre (HSC)
  - pre-encapsulated board, hermetic seal

- work centre: Adhesive Centre (AC)
  - board, adhesive, oven

6.2.3.3 Process Overview

We can refer to a microwave T/R process plan as a collection of manufacturing activities which can be performed to produce a microwave T/R module. This abstract notion may actually correspond to a set of detailed plans which can be utilised to fulfill manufacturing orders for this product. Each of these detailed plans are the result of a series of decisions concerning which particular method or operation to employ in order to satisfy a task. The process of moving from this initial, high-level task (i.e. making a T/R module) to a detailed plan, which can be executed by manufacturing workers, results in a hierarchical task network. The HTN provides the decompositional relationships from more complex, abstract descriptions to simpler, detailed instructions. We detail these possible operations and the requirements for them in the source scenario document [Polyak, 1998d]. Figure 6.4 relates the manufacturing operations from left to right with the abstract operations at the left and increasing detail to the right. The links in the figure represent possible decomposition relationships between the operations. For example, the "assembly" process is a decomposition of the more abstract "make board" process.

6.2.3.4 Concepts

This microwave T/R process plan domain requires the representation of a number of process and process-related concepts. These concepts are outlined in the following list. We provide an example for each concept. These concepts and examples are based on the requirements which are embodied in the source scenario description.

- **Activity Group**: A microwave T/R plan contains groups of activities which may have certain properties attached to them. One example of this is the total time calculation for the overall group (e.g. line "001 T 01", see Appendix entry E)

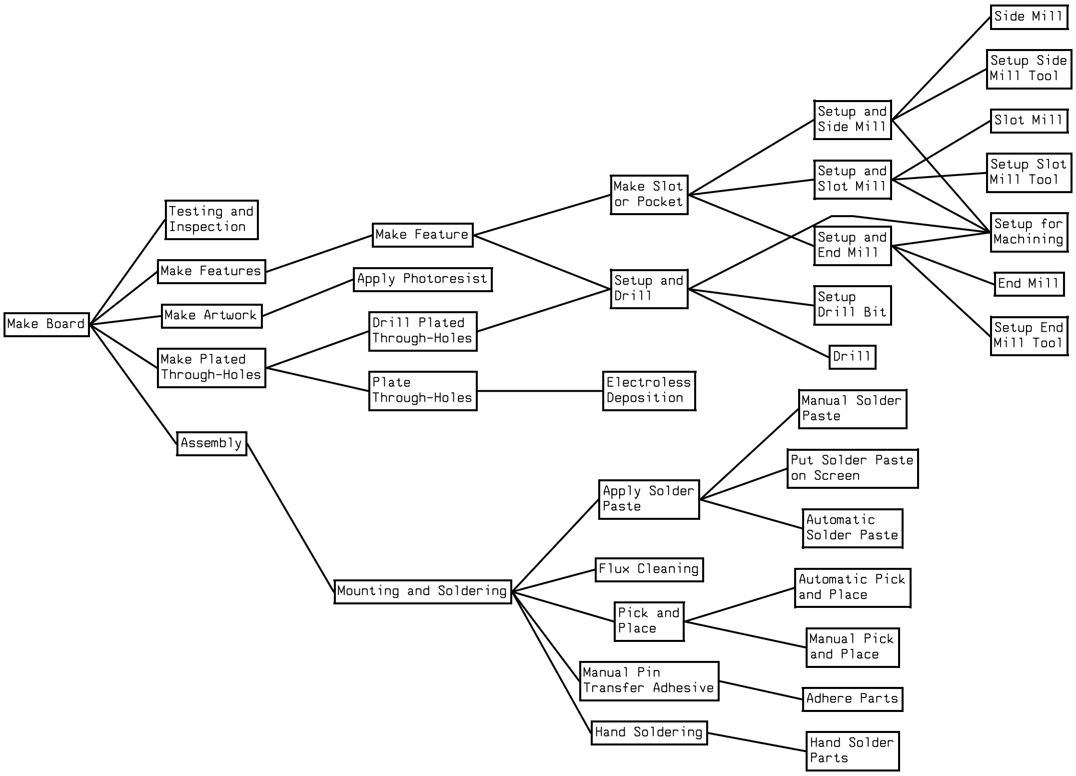

Figure 6.4: Microwave T/R module manufacturing processes

- **Annotations**: As a practical matter, microwave T/R plans may require annotations to be attached to them or to specific aspects of the plan. For example, the annotation "All time units are minutes unless otherwise stated" is a typical comment that may be attached to the top level of the plan.

- **Cost**: Cost is typically an evaluation of a microwave T/R plan that should be capable of being expressed. This applies to both overall and sub-plan costing. The calculation of this cost is material cost + setup labour cost + runtime labour cost + overhead cost: $\sum_i (M_i + LS_i + LR_i + OH_i)$

- **Design**: A microwave T/R process plan is the result of various electrical and mechanical design decisions along with process planning choices as evidenced in the integrated EDAPS module. An applied representation of a microwave T/R plan may need to be linked to its designs or design rationale.

- **Evaluations** In the cost concept, we pointed out that a financial plan analysis may be attached to a microwave T/R plan. Other plan analyses may include time to manufacture and product quality evaluations as well.

- **Filesystem** Some activities in the microwave T/R plan require a representation of filenames and possibly directory names as well. For example, line "004 C 02" specifies a photolithography activity which can find its input in the "real.iges" file. This may also be something required for attaching specialised documentation to a process as well (e.g. as mentioned in the design concepts section).

- **Events** Some activities may depend on the occurrence of particular events. For example, heat curing of the adhesive in "007 C 01" may begin when the event, "furnace comes to profile", occurs. This particular activity also provides an example of a representation of information pertaining to the estimated occurrence of this event (90-120 minutes).

- **Hierarchical Decomposition / Primitivity**: All throughout the detailed process section there are examples of both abstract and primitive activities. These activities form a hierarchical task network (HTN). This information should be retained in the representation of a plan for subsequent use in modular presentations of the plan, possible future replanning, etc.

- **Measurements**: A number of measurements are required at various points in the microwave T/R plan. These measurements include: depth, width, diameter, length and temperature. Another example of a measurement is a time interval. This includes things like: lead time, run time, setup time, subtotal group time, etc.

- **Measurement scale**: Along with a measurement, there should be some notion of measurement scale. While this may be implemented via an annotation, as was described in the annotation concepts section, it may also be necessary to have an explicit construct for scale to facilitate such things as translation of measurements between scales. This includes scales such as: Celsius, Fahrenheit, inches, minutes, etc.

- **Material constraints**: Some specialised constraints in the microwave T/R plan include material constraints. For example, a plan may require the ground plane to be copper or the dielectric to be Teflon.

- **Mechanical constraints**: Other specialised constraints may involve mechanical properties of materials such as a maximum board temperature or size constraints. Activities in the plan may depend on these mechanical constraints.

- **Operator**: An operator, worker or employee is required for all of the microwave T/R plan activities. A worker may be required to have a particular capability which qualifies them to perform an associated task.

- **Parameters**: Various activities in the microwave T/R plan are parameterised. The execution of the activity will be based on the value of some particular parameter such as location (for things like drilling, placing components, paste, etc).

- **Parts**: The microwave T/R plan should be able to explicitly represent parts such as components installed on the artwork. Various properties of the components affect considerations on things like time needed to manufacture and cost. For example, the number of leads on a component may determine its time to manually solder.

- **Position**: Various parts of the plan use abstract references to positions on the board. This may be required as a parameter for various activities such as drilling, placing components, etc. These positions may be expressed as values. Some examples in the sample plan include references at "001 A 03" to a datum point and bullseye.

- **Product**: As with the parts, there may be a need to access domain-specific properties of the product (in this case the particular microwave T/R module) in order to decide which activities are appropriate in manufacturing the product. For example, activities may depend on particular coupler gaps and dimensional tolerance properties.

- **Process/Activity**: At the heart of this process plan is a representation of the activities, steps, methods, or operations that are enacted to complete this plan. Examples of numerous constraints on the activities which may be performed have been illustrated (e.g. ultrasonic flux cleaning cannot be performed on integrated circuits). In the "operator" concept section there is a reference to the "performer" of the activity. Activities must also list which materials and tools they require as well as effects they may provide (e.g. "plated through-holes drilled"). They may contain a lead time, setup time, and run time which is described in more detail in the temporal concepts section. There may be a need to attach a particular manufacturability evaluation to an activity. These activities may also need to be numbered for reference in various presentations.

- **Quality**: A particular microwave T/R plan may have an associated quality evaluation. This may include some calculation of the yield such as: Quality (yield) $= \prod_i (Y_i)$, Y_i = yield (% of times within tolerance)

- **Ranges**: Along with measurements there may also be a need to express particular ranges of measurements. For example, the estimation of a time range from 90-120 minutes or a valid temperature range of an oven from 125-150 C.

- **Rationale**: There are several aspects of a microwave T/R plan that may fall under the general heading of rationale. This includes information about various causalities, dependencies and decisions in the plan. Causal relationships may communicate the reason why a particular activity is in the plan. For example, the reason we spray the board with vapour in the sample plan is to clean the excess flux. Certain activities may depend on other activities or, as mentioned in the product and parts concepts, on various properties of objects. The reason why a particular method was decided on over another one may be explained by some particular criteria (e.g. manufacturing time, plan quality, etc.)

- **Requirement**: Plans are shaped by the particular requirements and preferences that were specified. For the microwave T/R plan this may involve a listing of various components, locations, etc. along with constraints on the results of various plan analysis evaluations (e.g. cost, time, etc.)

- **Resources - Requires/Uses/Produces/Consumes**: As was mentioned in the Process/Activity concepts section, various objects play particular roles in the enactment of activities. These objects include materials, operators, work centres, machines/tools, work items, etc. A detailed semantics which specifies how the object is affected by the process will help to precisely characterise these roles.

- **Task/Objective**: A microwave T/R plan may also need to be related to a particular task or objective when it is being used to accomplish something. For example, "Make Board" may be a general task or objective for this sample plan.
- **Temporal Concepts**: The precedence relationships for the activities in this plan are all totally ordered. This may be expressed in PSL as relationships between activities, timepoints or intervals depending on the particular ontology selected for the PSL core. In CPF we use timepoints and cpo-ordering-constraints. In terms of duration, it should be noted that the values for setup times and run times may be the result of calculations, may depend on properties of components, and may simply be an absolute value.

6.2.3.5 Applying CPF

As we stated in Section 6.2.3, our focus here was to examine the *expression* of a transmit/relay manufacturing process plan in CPL as well as its *presentation* in CPE. Looking at the listing of the concepts in the previous section it should be obvious that a number of extensions to the underlying ontology (see Section 3.3.1) were required. As with the supply chain scenario in Section 6.2.2 we packaged these extensions in a simple, single microwave-module-extension. We noted as well that many of the identified concepts were already present in the CPO core (e.g. temporal constraints, process/activities, etc.).

The extension contains simple cases of new constraint types (e.g. material constraint, duration constraint, positional constraint) along with their legal expression, specific agent types (e.g. worker, operator), as well as activity-relatable-objects (e.g. machines, tools, work items). While it would be beneficial to give rich definitions for each of these new classes (especially for supporting translation), this was beyond the scope of this scenario work. It is important to notice though that this CPF approach provides the hook for attaching these clarifying, defining axioms or perhaps tying the concepts directly to existing external ontologies.

We investigated the import and user display of the hierarchical arrangement of the manufacturing steps in the Common Process Editor (CPE). The presentation of the process knowledge expressed in the CPL + microwave-module-extension served as an example to illustrate the use of the "other" constraints tab in the multi-process panel property sheet. In the CPL parsing engine (See 5.6) the CPL Spec module handles all "non-core" constraint types it encounters by posting them to the "other" constraints display. Currently CPE only allows free text editing of such constraints (as opposed to plug-in constraint editors). In the future though we will add constraint-specific plug-in support for non-core constraints by using reflection to match-up stored constraint-types read in to the appropriate editing methods (if they exist in the defined module).

In this scenario we also utilised the CPA to introduce temporal errors during editing sessions and to examine the result of the feedback provided by the tool. This facet of CPF support for managing process knowledge was presented in Section 5.4. We noted that a more feature rich temporal analysis tool would be required for realistic manufacturing processes. For example, CPA doesn't properly address processes with node references, thus allowing us to evaluate a complete plan or process.

During our exploration of this domain, we basically provided our own, user specified, "hand-translation" from the Northrup Gumman process plan (NGPP) expression (i.e. Appendix E) to CPL, compensating with extensions where needed. Given this knowledge, and our new extension, we could create a new bi-directional semi-automatic translation module (see Section 5.5) from (NGPP<->CPL) if it was required.

As the actual CPL output is rather lengthy, the CPL expression of the new source microwave T/R plan listed in Appendix E along with the defined microwave-module-extension appears on the project web area[6]. In addition to this we provide example presentations of this knowledge using CPE.

6.2.4 Camile Manufacturing Interoperability

This section describes the Camile manufacturing interoperability scenario which we use to illustrate an application of our thesis approach which structures the synthesis and management of process knowledge for manufacturing

[6] The CPF project web area is http://www.aiai.ed.ac.uk/~oplan/cpf

a commercial electronic product. We begin with a description of the domain followed by a review of the modelling concepts required. We then present a review of the products produced for this scenario.

During our work on phase three of NIST's PSL project, we developed a manufacturing interoperability scenario [Polyak and Aitken, 1998] in addition to the scenario we presented in Section 6.2.3. The microwave T/R scenario in Section 6.2.3 was focused on the *representation* of a process plan while the scenario presented in this section was written to show how a well-defined process framework could be used to improve the *synthesis and management* of a manufacturing process plan. In this way, the Camile scenario is more like those presented in Sections 6.2.1 and 6.2.2 with its grounding in a manufacturing context. The framework was envisioned as being able to support the communication or *interoperability* of process knowledge in a manufacturing environment. Specifically we pointed toward a typical manufacturing exchange in which knowledge from a process planner needs to be communicated to a job shop scheduler. The following text illustrates the manufacturing context in which this exchange occurs

> "A manufacturing firm makes a number of products. Associated with each product is a unique part number and one or more process plans. One process plan may be the preferred alternative, but the other alternatives are equally valid and yield acceptable parts. The process plan has a list of operations that must be performed in that sequence. The process plan has certain information about each operation: the operation name, the required resources, the planned run time, the planned setup time, the various parameters, etc.
>
> At a given point in time, the firm has a set of customer orders. Each order has a due date and requests a certain number of some parts. Completing the order means completing every operation in some process plan for that product.
>
> The production scheduling person must schedule the shop. That is, this person must create a schedule that states, for each customer order and each necessary operation, the workcenter that should perform the opera-

> tion, when the operation should begin, and when it should complete. The schedule is given to the manufacturing employees, who attempt to follow the schedule."

Thus, to an extent, this scenario subsumes the process plan representation in Section 6.2.3. This scenario though widens the scope to consider the organisational context of these processes and motivates the translation of process knowledge to various tools involved in the management or application of this information.

The product, process, and factory knowledge presented in this scenario has been taken from a planning and control reference case study developed by members of the Waterloo Management of Integrated Manufacturing Systems (WATMIMS) Research Group in 1991 entitled "Intelligent Manufacturing Management Program Sate of the Art Scheduling Survey" [McKay and Moore, 1991]. In this work, the authors provide a detailed account of a fictitious factory, the Camile Motor Works (CMW). Each characteristic and concept documented for this factory is based on factors which have been observed to be critical for decision making (predictive or reactive) in one or more "real" factories.

This case study covers normal manufacturing planning and control activities from the time of work order acceptance, including due date negotiation, to the time when final products are stored or shipped. CMW is described as a factory which produces a line of scale model automobiles which are constructed using

- a number of purchased parts used during assembly
- purchased parts which have additional internal processing (assembling, painting, etc.)
- internally fabricated components transformed from raw materials (various bodies, name plates, gear shifts, frames, engine block, spark plug wires, tyres, rims, etc.)

CMW has a wide variety of manufacturing processes at the factory and can be considered a hybrid environment. There are eight major departments comprised of 43 primary resources. The factory employs approximately 50 direct labour personnel which operate the plant and maintain the resources.

6.2.4.1 Product Overview

While three products, all variations of model cars, are presented in the source factory description (GT-200, GT-250, GT-350), we felt that it was sufficient for this interoperability scenario to restrict the scope to only address the most complex of the three. The GT-350 is described as a sophisticated running scale model automobile that is marketed via direct mail to company executives with optional license plates (various types to choose from). The product can be further personalised with the buyer's choice of 2 to 6 characters placed on an ID plate. The model comes in red or white. This high priced item is not stocked and is built only when firm orders are given. Delivery times of 4 to 6 weeks are indicated to buyers.

The product structure for the GT-350 is shown in table 6.2. The indenting of names is used to indicate the part sub-component structure. An unique part number is assigned to each component. The method of manufacturing for each component is listed in the right-most column. Some parts are manufactured internally or are assembled from other parts, while others are purchased or sub-contracted externally.

6.2.4.2 Process Overview

We can view the various departmental processes described in the scenario document as providing possible operators for a high-level collection of activities which are enacted to create a GT-350 product. As described in the GT-350 product structure (table 6.2), subcomponents of this product are either purchased, sub-contracted, or made internally. These process descriptions address the activities performed to manufacture the internal subcomponents. This top-down view of the manufacturing process provides an overall picture from an abstract, "make GT350" activity which is expanded down to the detailed departmental levels.

As Figure 6.5 shows, the GT-350 manufacturing process is divided into 6 main areas of work. The first five: make interior, make drive, make trim, make engine and make chassis are all unordered with respect to each other but they must all be completed before final assembly takes place. This figure uses an IDEF3 presentation. The main boxes indicate "units of behaviour" along with "and" node junctions. The dashed lines

GT-350 Product Structure			
Part	#	Part Number	Method
chassis	1	350–CHASSIS	assem
doors	2	x50–DOORS	purch
unibody	1	x50–BODY	intern
frame	1	350–FRAME	intern
engine	1	350-ENGINE	assem
block	1	350–BLOCK	intern
harness	1	350–HARNESS	intern
wires	1	350–WIRE-SET	intern
pwire	12	350–WIRE	intern
pwire	1	x50–WIRE	intern
drive	1	350–DRIVE	assem
motor	1	350–MOTOR	purch
electr.	1	350–PCB	intern
interior	1	350–INTERIOR	assem
cockpit	1	350–COCKPIT	subcon
dash	1	350–DASH	purch
lights	2	350–LIGHTS	purch
seats	1	350–SEATS	purch
gshift	1	x50–GEAR	intern
trim	1	350–TRIM	assem
options	1	350–OPTIONS	assem
licpl	2	350–LICENSE	intern
wheels	4	350–WHEELS	assem
tires	1	350–TIRES	intern
rims	1	350–RIMS	intern
decals	1	350–DECALS	purch
idpl	1	350–PLATE	intern
stand	1	350–STAND	intern

Table 6.2: GT-350 Product Structure Table

are used to indicate a decompositional relationship while the directed solid arcs indicate ordering relationships.

Each of these abstract activities are further detailed in the scenario document. This includes references to the departments which are responsible for the completion of the high level or detailed steps. Many of these steps involve work from more than one department.

6.2.4.3 Applying CPF

We have discussed the application of the complete series of CPF phases to a building and business scenario in Sections 6.2.1 and 6.2.2 respectively. In the work reported on in this section we have extended this application to a manufacturing scenario as well. As in the others, we followed the CPF phases presented in Section 3.1.1 which

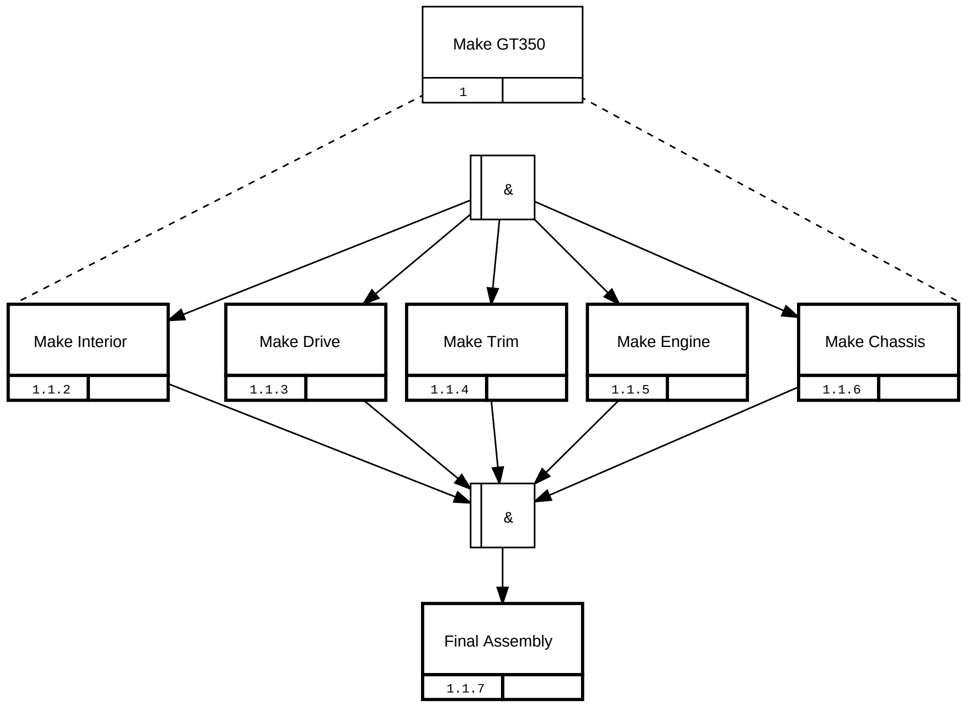

Figure 6.5: Top level process for manufacturing a GT-350

correspond to **requirements analysis**, **detailed domain development**, **process synthesis**, and **process management**.

During **requirements analysis** we dissected the structure of the Camile Motor Works with a focus on the GT-350 manufacturing processes. We found that the viewpoint analysis approach mapped very naturally onto the departmental structure of the factory (e.g. foundry, rubber works, machine shop, etc.). Additionally, there was a very clearly defined path for organising the domain into a series of hierarchical levels of detail.

Once again, our resultant set of CTDs were translated to an initial domain specification for the **detailed domain development** phase. During this phase we again encountered constructs which would require embedding contingency actions into the domain. For example the 350-PCB is an electronic unit which controls the functions of the 350 model. This unit must be tested in a process plan both by a "functional test" and a "stress test". In effect, we would like to design a process plan with sensing actions which, based on the outcome of the tests, might then branch into a fail/rework or passing handling action. As we are unable to embed such constructs in the current version of O-Plan TF [Tate *et al.*, 1994a], we simplified by eliminating the failure branch and added a new "failure-constraint" which serves as an instruction in such a case. As we said in Section 6.2.3, the project web area[7] contains information on this scenario's representation and extensions (see Section 3.3.2.5).

During **process synthesis** we generated new GT-350 process plans based on the "make GT-350" task. Various configurations can be created by changing global information about the state of the factory (i.e. the detailed domain specification) or by changing the "order" information represented by the requirements expressed within the "make GT-350" task. It should be noted here though that this AI planning-based process synthesis is different than those reported in the AI planning literature [Nau *et al.*, 1995, Gupta *et al.*, 1998, Batchu *et al.*, 1995] (see also Section 2.3.8). With this approach we are working at a much higher level of manufacturing process abstraction. We know what each step (or operator, schema, etc.) can do in very coarse granularity. These other approaches break the manufacturing planning problem down into much finer granularity typically seeking to match detailed product "features" (e.g. defined in a CAD system) with detailed manufacturing effects (e.g. ability to produce holes, mill, lathe, etc.).

Given the toolset we developed for the **process management** stage we could indirectly visualise and inspect both the manufacturing steps as well as the manufacturing material flow for a particular process plan by examining the detailed list of process constraints. In general though we believe that CPE doesn't adequately present the material flow perspective. Other toolset notations, such as IDEF3's Object State Transitions (OST), have more appropriate presentations which focus primarily on conveying the changes or transitions of various material's state. Given the CPF approach though, this could be integrated by creating a CPT for an IDEF3_{ost} <−>CPL.

To fully realize this approach in an applied setting, we would create translators (see Section 5.5) to the job shop scheduler's tool language (e.g. ILOG Schedule [Pape, 1994]) and to the process planner's tool language (e.g. IDEF3 [Mayer *et al.*, 1992]). In fact, we were centrally involved in helping to define an identical set of translators for these targets to be used with NIST's Process Specification Language (PSL) (cf.

[7] The CPF project web area is http://www.aiai.ed.ac.uk/~oplan/cpf

[Polyak and Aitken, 1998]).

6.2.5 Military Processes

In December 1998, during the AIAI work on the U.S. Army ***military scenario*** *that was being developed as part of a small unit operations (SUO) O-Plan transition project, we investigated the potential application of the CPE in supporting the Army's Military Decision-Making Process (MDMP). We report on this analysis as well the application of CPE to the ACP3 Air Campaign Planning Process Panel approach for ARPI TIE 97-1 project.*

In many cases, organisations typically already have a methodology for synthesising and managing their processes. In scenarios like this we would like to know if CPF can integrate with the existing methods and offer some value in helping to improve the overall approach. In order to evaluate this, we participated in the early stages of the U.S. Army **military scenario** that was being developed as part of a small unit operations (SUO) O-Plan transition project. One of the open issues at the time was whether the participating planning systems were going to be planning/monitoring the overall decision-making process or only planning/monitoring the detailed operation orders. Initial indications pointed toward O-Plan's involvement largely in the former.

We investigated the U.S. Army's approach to synthesising and managing their processes as defined in the Military Decision-Making Process (MDMP) found in the US Army Field Manual FM-101-5 [U.S. Army, 1997]. To simplify a bit, this process outlines the development of a set of various courses of action (COAs) from which the ultimate order approval is made. We envisioned the possible support provided from a subset of CPF as

- Displaying the overall MDMP process enactment instance in a hierarchical, network-style fashion
- Informing the organisational agent (e.g. Commander) of the current status of the MDMP actions and of updates to that information
- Receiving knowledge of newly synthesised COAs

- Browsing support for the COAs including an ability to flip between each hierarchical, network-style presentation
- Providing a bridge to other tools (e.g. evaluation tools)
- Saving individual COAs to filestore

The main component from CPF which could help to support this is the Common Process Editor (CPE). The CPE could facilitate the definition of the MDMP enactment instance and the tracking of the action status (e.g. using various node colours to indicate state) on a main process tab (or process worksheet). An enhancement (which we will discuss a bit more in the ACP3 work) could be added to provide a KQML (see Section 2.3.4.1) listener for such events. Specialised KQML events could be defined whose content carried a CPL process specification which would then be displayed on a separate COA process worksheet tab. Unfortunately, this approach was not utilised in the SUO transition project, but it did serve as a reasonable scenario for considering ways that the CPF components could be used when defined process engineering approaches already exist in an organisation.

This envisioned application of the CPF components is actually an extension of the ideas developed in the ACP3 Air Campaign Planning Process Panel approach for the DARPA/Air Force Research Laboratory (Rome) Planning Initiative (ARPI) TIE 97-1 (cf. [Aitken and Tate, 1997, Tate *et al.*, 1998a]). In this work, an early version of CPE was used alongside a COA evaluation matrix to provide a demonstration of the network-style presentation we mentioned above. The network was envisioned as showing the status of the decision-making process which was being updated by a KQML listener. [Aitken and Tate, 1997] outlines the KQML/MPA messages and the corresponding model state changes to support this plan/process creation.

6.3 Requirements Analysis

Back in Section 1.4 we discussed that at the outset of our efforts we had compiled a set of requirements with which to both guide and evaluate our work. This was seen as a complement to the work which we would perform on the applied scenarios that we have been discussing in Sections 6.2.1 *to 6.2.5. The focus of these requirements are mainly centred on the functional and representational issues involved with the shared CPF interlingua. In this section, we report on the groupings of these requirements which we use to examine some of the strengths and weaknesses of CPL.*

At the first steering group meeting of the Shared Planning and Activity Representation (SPAR) [Tate, 1998] on September 24, 1997 in Washington D.C. we presented our set of functional and representational requirements for a shared process/plan representation which we compiled from numerous sources [Polyak, 1997b]. These sources include work on process representations (e.g. PSL, PIF), DARPA/Rome Laboratory Planning Initiative (ARPI) plan, process, and schedule representations (e.g. KRSL, KRSL-Plan, CPR, SRI's Act, <I-N-OVA>, O-Plan TF, OZONE scheduling ontology, etc.), as well as numerous ARPI member's inputs (e.g. researchers, program managers, etc.) from various workshops.

This set of requirements assembled from these sources were expressed in simple, natural language statements. Somerville and Sawyer commented on the typical form for expressing such requirements

> "There is no best way to write requirements. It depends on normal organisational practice and the notations which are used by writers and readers of the requirements. ... Most requirements are written as natural language sentences supplemented with diagrams and tables of detailed information[Sommerville and Sawyer, 1997]."

These requirements were separated into representation and functional groupings. They were then clustered around various concepts and process/planning uses. The various clusters are not necessarily meant to be exhaustive nor mutually exclusive but rather as a structured conceptual map (i.e. they are simply a potentially useful way of partitioning the set into related categories). The categories are presented in this section. We will mainly be referring to "categories of requirements" as there are roughly 350 unique requirements in the complete set (See [Polyak, 1997b] or the CPF project homepage for a listing of the complete set). After presenting each category, we briefly consider the ways that CPL and, more generally, the CPF approach can be used to address these issues.

6.3.1 Representational Requirements

The representational requirements define the elements that are needed to express plan/process representations, either explicitly or implicitly. The requirements for these elements have been clustered into conceptual groupings. These groupings have been arranged in alphabetical order and are summarised below.

- **Activities**

  The representation of activities (actions, events, operations ...) is at the heart of plan representations. Activity specifications define the result of steps performed within a plan. Several requirements for the representation of activities can be traced back to the early work on the STRIPS planning system [Fikes and Nilsson, 1971]. Typical activity requirements usually involve some mechanism for abstraction and level decomposition as well.

  - **Applying CPF:** Using CPL we can specify action nodes which are associated with process specifications via node constraints. These nodes can be referenced in other detailed constraints such as input/output constraints which describe action conditions and effects.

- **Agents**

  The term "agent" in a plan representation generally refers to the people and systems assuming various roles throughout a plan lifecycle. Agents may be assigned as a performer of an activity, they may hold particular purposes, preferences, assumptions, etc. Agents may also be characterised by certain sets of capabilities that restrict what they can and cannot do. Agent models are sometimes necessary to clarify how the various agents interrelate.

  - **Applying CPF:** CPF identifies a specific sort for agents which can be assigned as performers of actions or processes. Extensions are required to express both capabilities or specific relations which could be assigned between agent instances (e.g. manages(a,b), supplier-of(a,b))

- **Control Structures/Execution/Simulation**

  Execution and simulation requirements emphasize the need for detailed control structures that can be embedded into the plan representation. This may include constructs such as loops, conditional activities, and world state monitors or queries. Reactive execution agents tend to operate from event-driven behaviour and require descriptions of procedures that describe how to respond to various occurrences.

  - **Applying CPF:** In several of the CPF scenarios we discussed the need for embedding contingency constructs in a process. This can be accomplished by creating a new constraint type which represents a conditional control structure. Knowledge of specific control structure extensions are required by any translator author who wishes to map this knowledge into a language which can work with these constructs.

- **Domain Knowledge**

  Plans are developed with respect to knowledge of a particular domain. This domain knowledge is typically expressed via the activities (or operators, schemas, Acts) that are present in the domain. Additional requirements usually involve information that can be applied globally across the domain. This may require the ability to state things that are always true, as well as the ability to infer truth from sets of domain rules.

  - **Applying CPF:** As we have seen, a major part of the CPF architecture involves the development of a detailed process domain (i.e. CPD files). The processes in the detailed domain specification act as building blocks for synthesising new organisational processes. The always constraints in the domain specification can be used to express simple pattern/value relationships or more complex domain rules, depending on the definition of the always constraint expression.

- **Evaluations**

  A number of plans may satisfy the requirements for a specific set of objectives. It is often the case that these plans need to be evaluated in certain ways to determine if they are acceptable for use (e.g. execution). Various domains require particular sets of criteria that can be applied for these evaluations.

  - **Applying CPF:** The concept of a process/plan evaluation can mean many different things for different domains. CPF provides some of the basic support for evaluations which determine whether certain fluents are true at some particular point in time. Extensions can be made for both including information that is to be evaluated or for storing the result of evaluations.

- **General Structures**

  General representational structures such as lists, sets, numbers, symbols, sentences, etc. are necessary to express plan knowledge. Particular structures may also be required to support specialised uses of this information (e.g. constraints for constraint solving, fuzzy rules for uncertain reasoning, etc.)

  - **Applying CPF:** CPF provides some generic data structures for things like sets, numbers, and strings. Extensions can be defined to create specialised structures for a domain. As much of the representation is similar to the work on PIF, PSL, and SPAR it is possible that extensions for structures or solutions from these efforts to concepts such as uncertainty and imprecision could be incorporated in CPF as well.

- **Goals, Requirements, Objectives, Mission, Tasks (GROMT)**

  A rich set of goals, requirements, etc. are usually needed to express tasks and to connect plans to their intended purposes. The state of a goal (e.g. satisfied, unsatisfied) the type of goal (e.g. achievement, maintenance, etc.), as well as expressed task constraints are used throughout the planning and execution process to provide purposeful behaviour.

  - **Applying CPF:** Various requirements for goal expressions exist in different domains. In CPF we have objective-specifications which can be used to provide state-based goals by placing input constraints on specific timepoints. We can also require particular actions to be performed or not performed. Efforts such as [Valente *et al.*, 1996] help by showing what different types of goal expressions are typically encountered or required. New constraint types can be added to capture these variations.

- **Organisational**

  Plans constructed within an organisational environment require representation of specific organisational concepts. This may include models of authority (permission to work with various parts of a plan), deadlines, milestones, policies, products, etc. The plans may be utilised by a number of systems within the organisation (e.g. workflow engine, process editor, project management software, etc.) which may place additional representational needs on the plan (e.g. particular views or filters).

  - **Applying CPF:** The utilisation of plans and processes throughout various systems in the environment can be partially facilitated by constructing translators between the source/target languages and CPL. The core CPF elements can be extended by incorporating work from various enterprise modelling efforts such as [Fraser and Tate, 1995, Uschold *et al.*, 1998] and [Fox *et al.*, 1993].

- **Plans/Schedule**

  A number of general requirements may also be expressed for the overall structure of plans or schedules. This may include the ability to represent various abstraction levels, sub-plans, decompositions, etc. Practical structural elements like document references and notes are typical as well. Shared plans structures may also require a mechanism (or set of mechanisms) to extend a representation for a particular domain or use.

  - **Applying CPF:** CPF provides the bare plan structure essentials for concepts such as abstraction, sub-processes/plans, activity specifications, etc. Specialised constraints can be created for items such as document references.

Additionally the annotation constraints can be used to attach notes to any of the CPF sort items.

- **Rationale**

  A rich plan representation may be required that not only contains the solution object (i.e. the actions, and orderings) but also a trace of how it had arrived at this result (see Chapter 4). This is analogous to the way that the proof steps of a theorem are required products of a proof. This may list items like the decisions taken, the alternatives considered, and the criteria used to evaluate the alternatives.

  - **Applying CPF:** This was covered in detail in Chapter 4. While CPF core doesn't include the specific design space analysis approach which we adapted, it can be expressed with the elements from the rationale extension we created. Other approaches to rationale (e.g. DRL [Lee, 1990]) could be substituted in its place.

- **Resources/Objects**

  Plans can create, utilise, manipulate, and destroy objects. These objects may play the role of resources that are used to complete a task. Resources typically need to be categorised by the way that they can and cannot be utilised (i.e. consumable, reusable, etc.) Detailed models of resources/objects and how they interact with activities may be required as well.

  - **Applying CPF:** CPF identifies a separate category for resource constraints. These constraints can be used to relate some action or process with activity-relatable objects. The expression of resource constraints can be tailored for specific applications of the framework. During our work with O-Plan we recognised that extensions were required to help with the mapping to Task Formalism. New sorts such as resource-unit or functions such as "object.resource-type" helped to capture the TF notion of resource.

- **States**

State representation is a very important part of plan representation. Plan generators must be able to create numerous hypothetical states that correspond to projected results of plan steps. Plan executors must be able to maintain a state representation that closely matches their current environment. States may represent the total set of concepts known at a particular point in time or may represent a difference between two such states.

  - **Applying CPF:** CPF has a very simple model for expressing the fact that some expression has some particular value at some point in time. We have worked with plug-in extensions to the ontology which provides an extended situation calculus and a theory for complex actions based on the work from [Pinto and Reiter, 1993a, Pinto and Reiter, 1993b, Gruninger and Pinto, 1995]. This allows us to map CPL process instances into a logical domain theory which can be used to answer queries on the value of expressions at arbitrary points along a discrete timeline.

- **Time/Space**

  Plans are usually constrained by time and space. Temporal constraints require plan steps to be ordered. This ordering may involve relative temporal relationships (e.g. A must occur before B), as well as various constraints on a single activity, such as activity durations. The Allen relations define a number of possible temporal relationships. Plan steps may also be required to be connected to metric temporal constraints (e.g. do A on July 28th at 8:00am). Spatial requirements can also be necessary to describe where an activity is performed.

  - **Applying CPF:** In fact, CPF considers the space of behaviour to be constrained by a whole range of constraint types. It is true that space and time tend to be major players in this set. CPF provides some basic temporal (or ordering) support which is slightly more expressive than the Allen relations. Metric constraints as well as relationships such as "not-between" require an extension. Various spatial formalisms are available which can also be plugged-in to address spatial relationships.

- **Uncertainty/Ambiguity**

  In real-world scenarios, it is often the case that one is unable to deterministically specify the effects of an action or to accurately hypothesise a future world state. In cases such as this, requirements are often made for ways to represent parts of the plan that are uncertain or ambiguous. This may include ranges on values (e.g. time windows, resource levels), abstract plan steps, or probabilistic estimations.

  - **Applying CPF:** The representation of uncertainty and imprecision in an action representation is still a very active field of research. As we said in the general structures section it is possible that solutions from other research can be snapped into the structure of CPF through some uncertainty extension (e.g. imprecise values may have an associated fuzzy-set). CPF does not attempt to provide a fixed solution to this issue.

6.3.2 Functional Requirements

The functional requirements define some of the intended uses of a rich, shared plan representation. These uses have been clustered into various categories. Many of these categories overlap in their functionalities but they have been listed separately in order to provide a more balanced presentation of the requirements. These categories have been arranged in alphabetical order and are summarised below.

- **Communicate Plan**

  Plans must be communicated to a wide range of agents (i.e. systems and people). Each of these agents typically require different presentations of the plan knowledge. Sometimes plan communication will involve the provision of specific views (e.g. hiding details, showing only relevant sub-components, etc.) as well as the presentation of different view types (e.g. state chart, PERT chart, etc.)

  - **Applying CPF:** The definition of an approach towards communicating plan/process knowledge is one of the underlying research opportunities which we identified in Section 1.1.3. CPF adopts a translation-based perspective around a shared, common interlingua. Various presentations of the shared knowledge can be made by developing mappings into appropriate notations or languages.

- **Domain Building**

  Domains building involves the acquisition and encoding of plan domain information which will be used to provide a model of the world. Domain editors should be able to express the entities and relationships that are specific for the domain and have the capability to constrain what things are possible in the world.

  - **Applying CPF:** CPF views most of the acquisition requirement as being addressed by the requirements analysis phase which is supported by the Common Process Methodology and toolset. The encoding requirement is viewed as the detailed domain development phase. This phase is supported by the Common Domain Editor. New entities, concepts and relationships can be added as extensions to the core process ontology.

- **Organisational Support**

  In an organisation, several agents will contribute to and evaluate knowledge contained within a plan. The plan should be amenable to organisational manipulation which could involve activities such as partitioning up parts of the plan and assigning them to various agents, user-specific browsing, etc. Various organisational concepts such as milestones, deadlines, annotations, etc. should be possible to access and view as well.

  - **Applying CPF:** Organisations can add various constructs for applying custom structure to process or domain specifications. For example, creating new functions for process sorts which could return the phase, level, or user with which they are associated. Likewise, various CPF sort objects can be related to extension sorts such as milestones, deadlines, etc.

- **Plan Editing/Browsing**

  Plan editing and browsing can impose very difficult requirements to address. Balancing what is possible for human planners to change and what is possible to change with system-based planners can get very tricky. Plan visualisation needs to be addressed effectively and efficiently. The solution should be scalable and appropriate to the task performed (e.g. simple operations should not require complicated plan editing actions.)

- **Applying CPF:** In CPF, the implementation of the Common Process Editor provides a basic example of an editing/browsing tool which can present the knowledge to human planners in an effective and efficient way. The current implementation of the integration between machine and human planners though does not truly support a mixed-initiative exchange (see Section 2.3.3). Possible future work could look at runtime communication between machine and human planners using CPL as the message content (e.g. add a process fragment, remove a process fragment).

- **Plan Execution**

  Plan execution requirements are usually very different than plan generation requirements. For instance, the plan representation may be required to contain points in the plan where the execution agent is instructed to obtain information from the environment or to synchronise parallel execution. Representation of execution progress and execution errors or exceptions should be inspectable.

  - **Applying CPF:** The CPF vision for plan execution is that a CPL process/plan specification can be translated into target execution languages (e.g. PRS [Georgeff *et al.*, 1989, Georgeff and Ingrand, 1989]). This points to our belief that while plan execution typically requires different representational elements, we believe that they can be handled with appropriate extensions. In this way we believe it is possible to unify representations for plan generation and execution similar to the way that Cypress [Wilkins and Myers, 1995] unifies both under the Act formalism (viz. SIPE-2 [Wilkins, 1984, Wilkins, 1988] input and PRS).

- **Plan/Schedule Generation**

Plan/Schedule generation requirements address those operations that are typically performed by systems or people while preparing a plan. This may require some of the "classic" AI planning and scheduling techniques (activity satisfaction, goal/activity ordering, etc.) Specialised techniques for plan generation may also be required as well (e.g. case-based planning, planning with uncertainty). The planning process may be required to be open, and inspectable throughout the entire operation.

  - **Applying CPF:** CPF supports automated plan/process generation by providing tools to build a plan domain in a structured way which can be translated into an appropriate format for an AI planner. Human planners can also build or edit processes and plans in the CPE as well. Stronger links between these automated and interactive approaches are currently being researched (cf. [Tate, 1997, Veloso, 1996]). The open, inspectable requirement on the planning process was one of our motivations for incorporating the design rationale approach into the framework.

- **Plan Evaluation/Critique**

Specific requirements may be used to address how the plan is evaluated or critiqued. These operations may be applied during plan generation or may be applied to a resultant "plan". Evaluations may be required to check things like consistency, robustness, goal achievement, risk, etc.

  - **Applying CPF:** The CPF approach envisions decoupled evaluation modules. Translation to these modules permits parts or all of the process specification to be evaluated in various ways. Extensions to the language allow these modules to embed the results of evaluations if required. To illustrate this decoupled module approach we selected a method for performing an evaluation of temporal consistency (see Section 5.4). In this case we communicated the results of an evaluation separately, but we could have easily created and embedded custom issue constraints based on the results of temporal evaluation.

- **Planning System Synthesis**

  Recent research in planning has pointed toward possible planning approaches which synthesise a planner from a specification of a planning problem [Srivastava *et al.*, 1997]. This approach may impose certain requirements on how to bias the search with the help of control knowledge acquired from the user.

  - **Applying CPF:** When we developed these requirements we envisioned the possibility that we could use a detailed domain specification to help infer which planning techniques would be most appropriate. Most of the domains that we worked with though in the course of the thesis work were very closely aligned with the HTN-style of planning. It is still possible though that future work with more varied planning problems could lead to such a link.

- **Task Assignment**

  Task assignment may need to be flexible and permit the expression of a variety of constraints on the problem to be solved. Task assignments may involve specifying various levels or phases, for instance. This may constraint parts of the plan to agents that have appropriate authority. Task assignment may also involve an expression of things that must be true (i.e. enforced) and things that are preferred to be true (desires or preferences).

  - **Applying CPF:** When working in CDE, processes can be flagged as taskable which facilitates their translation to task schemas in TF. Built-in or custom constraints can be placed on these taskable processes (e.g. include some action, achieve some condition at some timepoint, respect some custom constraint). CPF supports the expression of agent preferences or requirements.

6.4 Conclusion

In this chapter we discussed two separate sources of work in which we examined the CPF approach presented in this thesis. This includes work on the portfolio of process scenarios as well as the set of representational and *functional requirements. In this section we sum up what we have reported on here and discuss our conclusions.*

Throughout this chapter we have discussed both the scenarios and requirements which have served to influence the shape of this thesis work and to guide its development. We shall briefly summarise these items and consider some of the conclusions from this work as a precursor to our thesis conclusions which we present and explore in greater depth in Chapter 7.

We can characterise the scenarios which we have presented in this chapter with respect to the CPF approach in the following way

- **Building scenario**. Illustrated one complete pass through the CPF phases. Utilised extensions for CPF tools, TF elements, rationale extensions.

- **Business scenario**. Illustrated multiple passes through the CPF phases with a looping between detailed domain development, process synthesis, and process management. All extensions from the **building scenario** plus business (supply chain) extensions.

- **Manufacturing 1 scenario**. Illustrated the representation of a process plan. Used CPE for process visualisation. Utilised extensions for CPF tools and manufacturing extensions. Used CPA to evaluate temporal evaluations.

- **Manufacturing 2 scenario**. Illustrated one complete pass through the CPF phases. All extensions from the **building scenario** plus manufacturing (Camile factory) extensions.

- **Military scenario**. Considered applications of CPF for supporting organisations with predefined methodologies. Illustrated uses of CPE for managing and presenting multiple instances of process knowledge.

As we can see, the entire CPF set of phases has been applied to building, business, and manufacturing examples. Additionally we examined some support for process representation and presentation in military and manufacturing cases. Overall, these examples helped to validate the notion that, at least for this set of organisational processes, we could effectively apply the common elements from CPF. We also used these examples to evaluate what types of extensions might be required in specific cases. These examples also helped to show that further work is required, for example, to better understand how detailed domain specifications or even the initial requirements influence or are influenced by the capabilities of AI planning approaches (cf. [Cottam and Shadbolt, 1996]).

Breaking down the requirements we gathered into various categories gave us the opportunity to reflect back on both the design and implementation of CPF. For each of these clusters of requirements we considered ways that CPF can be employed to meet them. As we expected to see, a lot of these applications involve a "build an extension" solution. This is a direct result of the lessons we learned by working with other projects such as PIF, PSL, and SPAR which found that by adopting a smaller, well considered core you can gain greater flexibility. This position of remaining agnostic to such issues as the detailed expression of constraints (e.g. goals) can be found in other recent work as well (cf. [Valente *et al.*, 1996]).

In the following chapter we will structure and explore the lessons and implications which are connected both to our evaluations derived from this work with the scenarios and requirements and in our overall building and understanding of this integration framework.

Chapter 7

Conclusions and Implications

Using a jigsaw puzzle analogy, we can consider the role of this final thesis chapter. We began by presenting the research problem in Chapter 1 which introduced a set of pieces for a jigsaw puzzle. The literature review from Chapter 2 started to put the pieces together to uncover a picture, but showed that some pieces were missing and so the complete picture could not be shown. In Chapters 3, 4 and 5 we described the hunt for the missing pieces. Chapter 6 helped us to understand how the new pieces both fit together and partially fit into the existing puzzle. Finally this chapter briefly summarises what the picture looked like after Chapter 2 and then explains how the new pieces completely fit in to make the whole picture clear. In doing so, we will return to the research issues, hypotheses, and problem. We also consider both the limitations of this work and the need for further research.

7.1 Introduction

At the end of Chapter 2, we had outlined our knowledge of the existing work which either applies or could be applied to some aspects involved in the management of organisational process knowledge. While our focus is on an AI planning perspective, we pointed toward possible synergies with a number of research areas such as knowledge sharing (Section 2.7), logic-based representations (Section 2.4), design rationale (Section 2.5), requirements engineering (Section 2.11) and knowledge acquisition (Section 2.3.7). Some of these links were motivated by individual threads of research from the

AI planning community and also, more specifically, from the historical context of the research work here at The University of Edinburgh (Section 2.13).

One of the main tasks we were faced with involved articulating a coherent structure in which the integration of the various contributions could be realized to produce an effective approach towards managing organisational processes. We noted that AI planning has been applied to managing organisational process knowledge in the past, but that there remains a gap between the application and the research as well as a lack of repeatability of approach. In Chapter 3 we presented our solution as the Common Process Framework which sought to help fill that gap. This framework provides some of the missing pieces which we felt were required to clarify and coordinate an understanding of how these disciplines could be made to interrelate. These missing pieces were foreshadowed in Chapter 1 under the auspices of identified research issues and hypotheses.

In this chapter we will revisit these missing pieces by discussing each research issue and hypothesis. In Section 7.2 we describe our conclusions for each based on the implementations presented in this thesis work. Section 7.3 then steps back to consider the complete picture and discusses our conclusions for the overall research problem. These conclusions have implications for applied uses which we present in Section 7.4. During our work on this contribution we identified particular limitations which we consider in Section 7.5. We conclude with thoughts on further research in Section 7.6 and a synopsis in Section 7.7.

7.2 Conclusions on Research Issues and Hypotheses

In Section 1.1.3 and Section 1.2 we introduced the identified research opportunities and research hypotheses, respectively for this thesis work. These pieces of the research puzzle motivated our research and our search for an integrated solution. In this section we present these items along with our conclusions formed in pursuing them.

7.2.1 Research Issues

We begin our review of the conclusions of our work with the research opportunities or issues which we identified in Section 1.1.3. For each of these issues we discuss both our solution and some of the important contributions of the effort spent towards tackling it.

- How do organisations elicit requirements for world description knowledge?

This issue stems from our belief that the AI planning model offers significant benefits to organisational process management. Specifically, the construction of a world description from which new processes may be synthesised enables a certain level of automation to be introduced into the overall management approach. This contrasts with other typical approaches where human process modellers generate new process configurations by hand from scratch using simple notations such as IDEF3 or UML. The acquisition of this world description knowledge though is still a very active field of research (see Section 2.3.7).

Earlier work at Edinburgh noted the similarity between this research issue and work being done in requirements engineering (see Section 2.11). We explored and extended this work by pursuing the links to the updated version of the CORE methodology used in this work. We concluded that our adaptation of the CORE methodology (see Section 3.2) does aid in providing a principled, well-documented approach to eliciting the requirements for specific domains. We also concluded though that this requirements elicitation process is best viewed as spanning two distinct phases. Following the requirements analysis, a detailed domain development phase is required to make design decisions which live closer to the AI planning implementation. In Section 7.5 we discuss limitations of our implementation of the elicitation process.

- How are shared world and process/plan descriptions represented and communicated?

We concluded that processes are designed artifacts which are typically shared amongst agents and systems involved in the overall management of such knowledge. As we advocated an approach in which a world description is first created which aids in the subsequent synthesis or configuration of new process descriptions, we needed to address the knowledge sharing involved for both world and process descriptions. Both process knowledge representation 2.3.1 and knowledge sharing 2.7 are also areas in which a significant amount of research is being performed.

Based on a requirements analysis we conducted [Polyak and Tate, 1997], we concluded that the <I-N-OVA> perspective of a constraint model of activity would be capable of meeting our needs for this research issue. Following the work in [Tate, 1996d] we developed an ontology (CPO) with a specific set of classes, functions and relations with which to express this knowledge. Given these constructs we designed a specific language (CPL) to define instances of world and process descriptions. Finally, we provided examples of a translation-based approach to knowledge sharing in which mappings are built between source/target languages and the shared CPF interlingua. We concluded that this method is very effective and simple for many of the shared, core constructs but that the approach needed to be highly flexible and easily extended in order to scale to realistic processes such as those described in Chapter 6.

- How do we incorporate and support heterogeneous knowledge sources?

By this we meant that various agents and systems involved in the lifecycle of a process description (or world description) may produce a variety of related knowledge elements concerning the description which are expressed in their own way. For example, a cost analysis tool might produce an evaluation of the process cost or a temporal analysis tool might produce a list of temporal errors. We concluded that a reasonable approach would be to integrate these "related process knowledge elements" within the existing description (viz., expressing it in the CPL file). In order to do so we described a mechanism for extending the core ontology where necessary. This extension may then be required for some particular mapping/translation. So, for example, the cost analysis tool extension might involve defining a new cost constraint. The process cost from the tool would then be mapped into this new cost constraint during translation back to CPL. This solution draws on the PSV approach originally defined in [Malone and Lee, 1990] and used in the PIF and PSL projects.

- What process-related elements are common to most applications?

This research issue has a long pedigree which we acknowledged in Section 2.3.4.2. As we have indicated in this section, we chose to align our research with the core elements from the <I-N-OVA> constraint model of activity. To a certain degree, we pursued a partial validation of this model through our work with the scenarios and requirements which we presented in Chapter 6. We concluded that most of the <I-N-OVA> elements underlying the concrete Common Process Ontology were an adequate set for expressing the basic process design. We also found that most of the extensions required in the various scenarios could be reasonably packaged into extension modules, but that additional research is needed to better understand how to structure modules, locate, and control changes to them in a well-defined manner.

- How can we customise and extend the knowledge representation to address specialised needs?

We have already discussed our conclusions for a need to be able to extend the knowledge representation with respect to adding new classes, relations, or functions given specific applications. Additionally though our work has also concluded that there needs to be flexibility in the form of expressions. This concurs with the findings reported in [Swartout and Gil, 1996, Swartout and Gil, 1995, Valente *et al.*, 1996, Wickler, 1999]. In fact, CPF's approach to specifying new constraint types is similar to specifying new XML tag types [Holzner, 1998] (see page 96). Effectively adding a new constraint type instance bounded by the double quotes in the constraint.expression relation is analogous to an instance of XML tag bounded by open/closing tags. Using second-order expressions in Ontolingua [Gruber, 1993], an extension author can also describe valid grammar for the new expression type. The intention of this meta-knowledge is to provide translation module authors with the information they need to properly parse the new constraint type. If most of the translation modules for an application of the CPF were to be written in Java, it would perhaps be favourable to provide a JavaCC grammar specification (see Section 5.2.4 and also the role of JavaCC described in [Crow and Shadbolt, 1999]) which could be used to automatically generate appropriate sub-parsing modules.

- What type of tools are required to support this approach?

We identified the type of tools and their relationship to the phases which we have defined in the CPF architecture (see Figures 3.1 and 3.2). We concluded that a tool would typically be needed to support the requirements analysis methodology. We implemented this as the CPM toolset which provides users with simple consistency management checks, intuitive graphical browsing, and model translation to the shared interlingua. Throughout Chapter 5 we outlined support for visualising, editing, and exchanging both world and process/plan descriptions using the CDE and CPE internet tools. We concluded that these tools needed to be flexible to match the flexibility in the underlying representation (e.g. runtime plug-in constraint builders). We described a need for custom translation modules (e.g. to automatically synthesise a target plan domain representation) and also presented a sample implementation of an evaluation tool which can report interval-based critiques of CPF process knowledge.

Importantly though we concluded that the CPF approach does not require the use of all the tools or phases as we described in Section 6.2.5. The CPF architecture can be viewed as a toolbox from which components may be interleaved with other existing tools and methods. Also, we point out that as in [Drummond and Tate, 1992] there is value in simply applying the AI planning model and *representation* even if an AI planner is not actually used in the overall approach.

7.2.2 Research Hypotheses

We continue our review of the conclusions and contributions of this thesis by revisiting the research hypotheses which were presented in Section 1.2. These hypotheses were written at a relatively high level and are not necessarily subject to absolute acceptance or rejection but rather require a qualitative analysis with respect to this thesis work.

H1. *Organisational processes can be effectively represented with AI plan descriptions.*

H2. *Knowledge of available actions or potential process options can be effectively represented with AI world descriptions.*

Using the AI planning based representation which we adapted from [Tate, 1995, Tate, 1996c, Tate, 1996d] and developed in this thesis work, we created both domain models and example process configurations for diverse applications such as house building, supply chain coordination, manufacturing and military operations. We believe that this approach is much more effective than the simple action, ordering/control flow diagrams which are typically used to describe the relationships between actions in an organisation. By grounding new configurations of organisational processes in rich domain models (world descriptions) we can capture much more powerful notions such as resource commitments, state information (e.g. conditions and effects), dependencies, etc.

> **H3.** *Process-relatable objects and their relations can be effectively represented within AI world and plan descriptions.*

Based partially on our collaboration with the PIF and PSL projects and the thesis work described here, we concluded that an organisation's set of processes can be thought of a set of constraint configurations relating some specific collection of actions and objects. The specific relationship between actions and objects varies (e.g. requires, consumes, produces, etc.) and objects might be interrelated in some way (e.g. part-of). In both cases we can create extensions for specific applications which captures these specialised objects and relationships. We have worked on examples of this for such things as supply chain objects and manufacturing objects.

> **H4.** *Incomplete or completed designs of organisational processes can be shared amongst people or systems.*
>
> **H5.** *Incomplete or completed designs of world descriptions can be shared amongst people or systems.*

There are two major factors being considered here: incomplete designs and knowledge sharing. We have already discussed the latter in our stance on the translation-based approach to knowledge sharing in which translation modules map information into and out of CPL. The former factor concerns the ability to embed knowledge of the state of the design in the exchanged content. This aspect is a central tenet of the underlying representation used in this thesis: <I-N-OVA> . The "I" in <I-N-OVA> (or more generically the <I-N-CA> model [Tate *et al.*, 1999a]) stands for issues. Issues may contain plan flaws, unexamined analyses, or outstanding agenda entries which we can think of as workflow items which need to be processed in order to realize a "complete" design. In the expression of the design knowledge these simply appear as another type of constraint but they can be considered to be "future" or "implied" constraints on behaviour which may ultimately result in the addition of other detailed constraints (e.g. ordering constraints or include node constraints).

H6. *Rationale of a design can be shared amongst people or systems.*

We believe that such a thing is generically possible as evidenced by the work collected in [Moran and Carroll, 1996]. More specifically though, this hypothesis was aimed at the rationale underlying process designs. Our first step towards forming a conclusion on this issue was to identify what is meant by process rationale (see [Polyak and Tate, 1998]). We found that this category of knowledge encompasses causal relationships and dependencies as well as information defining the space of design decisions. While we noted that the expression of dependencies and causal information is a well-researched issue in the AI planning literature, there was comparatively less work on planning decisions. Using an approach from design rationale (see Chapter 4) we explored the types of questions/issues which are typically encountered in a classical AI planning design space. We concluded that this design space can be integrated with the constraints on the behaviour space by defining an extension for questions, options, and decision criteria.

H7. *The expression of design knowledge can be flexible in order to interoperate with a range of people or systems.*

We have discussed our conclusion that it is beneficial and in some cases necessary for the representation to be flexible and extendible (see Section 2.7.2) in order to support interoperability. This flexibility encompasses the form of constraint expressions as well as a mechanism for adding new classes, relations, and functions. This flexibility in representation though does not come without its costs. This places significant importance on the design and implementation of the translation modules and also influences the design and implementation of tools for visualising and managing this knowledge (i.e. CPE and CDE). This also raises issues of how to appropriately structure such extensions and how to deal with a whole range of issues such as importing conflicting constructs. We will discuss this further in Section 7.5.

H8. *A generic tool for visualising and editing organisational processes can be designed which addresses a range of process types.*

H9. *Generic tools for eliciting, visualising and editing world descriptions can be designed which address a range of world types.*

Hypothesis H8 corresponds to our work on the Common Process Editor (CPE) and its application to the various scenarios in Chapter 6. We concluded that the main presentation of CPE should focus on the actions and the orderings using a graphical node-arc style. Node expansions are presented in separate windows with a mechanism for navigating up and down in the process hierarchy. By bringing up the process "property sheet" a process user can view and edit the rich, detailed constraints. These constraints are separated into various categories (e.g. input, output, variable, etc.). While this approach encourages a common view of organisational processes and worked well for the scenarios we examined, we also believe that the translation approach is needed to tailor presentations to target languages (see Appendix E) which exist in specific environments (e.g. business, manufacturing, or WWW).

Hypothesis H9 corresponds to our work both on the CPM toolset and the Common Domain Editor (CDE). Using CPM we followed the requirements engineering process for eliciting the overall domain structure in the various scenarios. We utilised the same visual presentation for processes in CDE with the exception that commitments to specific node decompositions were not allowed. Additionally, the CDE presents the domain using a generic tree structure which: organises the domain actions into a set of domain levels; lists the domain objects (instances and subclassing); and contains the set of constraints which globally apply to the domain.

H10. *Efficient system-specific translation may be possible between source and target languages for those systems using shared models.*

Using the example from the manufacturing scenario in Section 6.2.4 we can see that knowledge of organisational processes are often present in organisations in a variety of formats for different purposes. In this specific case we might have a source IDEF3 presentation used in process plan modelling and a target ILOG [Pape, 1994] presentation for job shop scheduling. Underneath both of these presentations is a common, shared model of what comprises a manufacturing process plan. In our work, both on this thesis and in collaboration with the PSL project, we found that a translation based approach allows us to identify the common concepts which can be expressed in the shared interlingua. We can then establish translation modules which enable knowledge sharing in an way that facilitates the integration of additional language sources or targets.

7.3 Conclusions on the Overall Research Problem

In Section 1.2 we stated that the research problem addressed in this thesis can be described from two different viewpoints. We can look at this from an organisational management position or a technology solution position. This roughly corresponds to a top-down or bottom-up perspective. In this section we summarise how our work has been able to bridge the gap between these two perspectives and integrate an approach towards managing organisational process knowledge.

How can we improve the methodology of synthesising and managing organisational process knowledge?

In Section 2.12 we discussed the opinion that the value of an organisation stems from the knowledge of its own processes and in its ability to effectively manage that knowledge. Additionally we discussed some tools which have been developed to address aspects of this overall process management methodology in Section 2.12.1. Our opinion

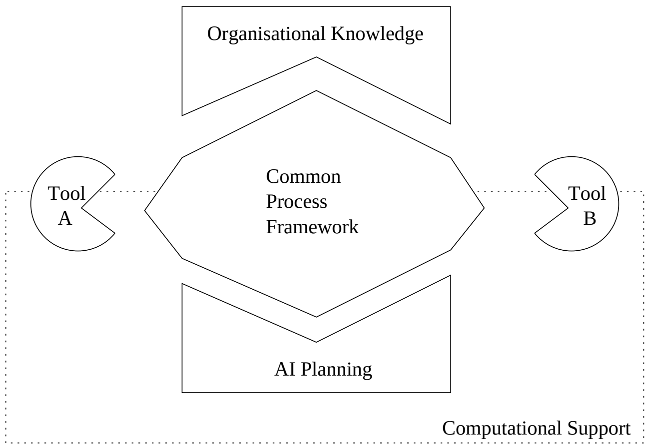

Figure 7.1: Integration Role of the Common Process Framework

was that in many of these examples the underlying representation was very weak, that they often used proprietary formats and that they represented islands of computational support. We felt that the methodology of synthesising and managing organisational process knowledge could benefit from work that articulates a coherent framework, supports a rich, knowledge-based declarative format and that would provide an avenue of integration for the tools involved. It has been our goal throughout this work to present such a framework.

The perspective of this problem statement can be considered to be anchored towards the top of Figure 7.1. This figure illustrates the notion that we have some organisational knowledge which we wish to interface with computational support. The role of the Common Process Framework is to interface with that knowledge, to provide some of the computational support, and to interface with other computational elements as well. In doing so, we concluded that CPF is able to benefit by shopping from various techniques from requirements and ontological engineering, design rationale, knowledge sharing, and especially AI planning.

How can organisational process design benefit from AI-based planning and plan representations?

Starting from the bottom of Figure 7.1, we can consider the problem tackled in this thesis to be one in which we investigated how AI planning and plan representations could interface with an approach towards managing organisational process knowledge. We concluded that the capabilities offered by AI Planning do not directly plug into organisational knowledge, but can be considered part of framework which provides the appropriate computational support. CPF provides the integration required to bring these aspects together. For example, we determined that at least two phases were required in acquiring and preparing the knowledge needed for an AI planning domain representation. CPF provides the support for building this "domain specification". We also found that these capabilities need to be integrated alongside the capabilities of other tools (e.g. tools A and B in Figure 7.1) involved in the process lifecycle. Our conclusion was to create an interlingua based on AI planning representations for exchanging knowledge about the organisation's domain. This interlingua was also used to express organisational process configurations constructed from the domain specification.

7.4 Implications for Applied Uses

In this section we consider some of the implications of this thesis work in relation to its potential applied uses. We discuss the type of organisation that can benefit from our approach an in doing so, we present a few more examples of situations in which the CPF could be applied.

The work described in this thesis documents an approach towards bringing research from AI planning and one of its possible applications, organisational process management, closer together. While we discussed various applied planning systems in Section 2.3.5 we noted that there was typically a lack of information surrounding the way that these systems "fit" into some organisational context. Our efforts have sought to provide a generic approach which can address this context issue for certain types of organisations.

As we stated back in the beginning of Chapter 1, there are many types of organisations and consequently process management can mean many different things. We have envisioned that the type of organisations which could benefit from our approach are those which have:

- a very action-centric view of process
- a domain which lends itself to discrete identification
- a need to repeatedly reconfigure some process or set of processes based on changing domain or task requirements
- processes which typically contain several levels of detail
- a requirement for interoperability amongst people and systems

In Chapter 6 we described our application of the CPF approach to some of the organisations which we believe to be good matches for this criteria. We will now present descriptions of a few more examples of such organisations and consider the way that our approach could be applied to improve the methodology of synthesising and managing organisational process knowledge.

7.4.1 Planning an External Audit

> Large accounting firms perform financial statement audits for a variety of companies in several industry segments and countries. The process of building an audit engagement plan is a complex process that exposes an accounting firm to a great deal of risk. Many of these risks can be managed by tailoring the plan to fit the particular constraints of an engagement. Activities are selected based on detailed company knowledge (e.g. revenues, ownership, capital, core industry, etc.), engagement information (e.g. length of audit, estimated fees, etc.), country laws and regulations, as well as "best practices" learned by the firm over time. The resultant plan serves to guide auditors through an engagement while minimising the potential risk. Ultimately a partner, and in some cases also a concurring or quality control partner, in the firm will be held responsible for the decision to accept the client and proceed with the plan. An externalisation of the plan must be provided for the partner's review. The plan will also be used to convey execution instructions to the audit engagement team.

We considered some of the work on planning a financial audit in Section 2.9. Using the organisational properties we identified we can see why this process management scenario is a good candidate for a possible future application of our work. Accounting firms are very action-centric and the domain is typically well-defined by laws and regulations which can be expressed as a set of individual detailed constraints on each potential step in an audit engagement plan. The process of designing a specific engagement configuration is something that is repeatedly performed by members of the organisation. These audit engagement plans need to be communicated to various people involved in the engagement and to tools which can perform such things as risk management assessments. CPF can provide the tools, methods, and techniques which applies a core AI planning representation and can enlist the capabilities of an AI planning system to help reduce the time it takes to select and organise the appropriate steps.

7.4.2 Designing a Product Strategy

> Advertising agencies that manage clients with large advertising budgets (10-100,000,000 US dollars/year) go through "planning seasons" to coordinate the activities of marketing and advertising a companies line of products or services. Various constraints are placed by the client, agency, and production company on the plan. For example, the ad agency needs to know: how much money is available for production and media, what is the objective of the message (e.g. penetration, or usage), when does it need to be "on" (e.g. Q1, Q2, etc.), how many people need to see it (Target Rating Points, TRPs), etc. Given these constraints, various planning decisions can be made (media type, approach, etc.) and activities can be selected and arranged in order to meet the goals of the client within the timeframe specified. The plan will then serve to guide the process of producing the advertising and marketing for that season.

A similar example in another domain is the advertising process management task of designing a product strategy. A product strategy is a sketch of the process which is going to be enacted, in part, by the client, agency and production company. Again, a product strategy tends to be very action-centric, identifying those actions that will be used to meet the client's need for the product. Ad agencies have fairly well-defined sets of discrete activity (e.g. for a print ad you must perform A, B, and C). A complete product strategy plan may include several details relating to cost, milestones, location, materials, etc. Advertising agencies typically spend significant amount of time repeatedly developing these highly detailed plans for each product which their clients wish to advertise. These plans need to be communicated across a range of people and systems. Again it is possible that an AI planning system could aid in synthesising these plans, but we believe that an approach such as the CPF is required to bring the appropriate level of organisational support for enabling the transference of technology to a commercial sector.

7.4.3 Managing Military and Workflow Processes

In Section 6.2.5 we discussed our investigation of using parts of CPF to support the Army's Military Decision-Making Process (MDMP) as part of a small unit operations (SUO) O-Plan transition project as well as the application of CPE to the ACP^3 Air Campaign Planning Process Panel approach for the ARPI TIE 97-1 project. We observe that military operation orders tend to be very action-centric and involve several discrete elements. Orders at a particular level in the command hierarchy tend to be very detailed. Orders are repeatedly generated by military organisations and need to be shared between various participants in the chain of command.

There are a number of other military applications which we think are appropriate for CPF or could benefit from the details of the approach which were presented in this thesis. For example, during the International Workshop on Knowledge-Based Planning for Coalition Forces (Edinburgh, May 1999) Northrup Fowler, from the U.S. Air Force Research Laboratory/IFT, outlined the vision of a "battle infosphere". The evolution of the air force technology outlined a four layer model including: visualisation and human-computer interaction; analysis tools and planning aides; information management; and networking/communications. Fowler commented that little or no research is currently being done on the information lifecycle and that the current analysis tools and models are stovepiped. He also outlined a need for "living plans" which he described as containing linked decision rationales as well as a requirement for being able to identify several senses of domain object instances (e.g. a bomb might represent manufactured item, ordinance, cargo, aerodynamic object, configuration item, financial property, etc.). Considering CPF we can see the following overlaps:

- CPE, CDE, and the CPM Toolset address parts of the "visualisation and human-computer interaction" layer
- Tools such as the CPA and the translation capabilities of CPF aide in supporting the "analysis tools and planning aides" layer and helps to partially overcome problems associated with stovepiped systems
- The interlingua and underlying ontology from CPF provide research that helps us to understand the "information management" layer
- The overall CPF architecture and our implementation of it contributes toward research on an information lifecycle
- Our work integrating process artifacts and process decision rationale contributes towards our understanding of "living plans"
- Grounding our representation in a sharable process ontology gives us the capability to define process objects in flexible ways that can capture their various roles in a domain

In addition to military organisations, we also view organisations that deal with workflow as potential targets for our continued work. Workflow domains and processes tend to also adhere to our model criteria. In fact, the Task-Based Process Management (TBPM) project, involving researchers from Loughborough University and the Artificial Intelligence Applications Institute (AIAI), have already begun plans to integrate elements of the Common Process Framework (CPF) into their work (cf. [Jarvis *et al.*, 1999]). The overall aim of the TBPM project is to support the management of change in business processes with the help of intelligent task management and coordination technologies (e.g. workflow).

7.5 Limitations

In this section we discuss the limitations of our work. We look at the sources of these limitations and consider ways that the limits may be overcome or simply point out how they influence the scope of applicability of this work. This is a lead-in to our discussion on future work in Section 7.6.

Some of the limitations of the work described in this thesis were a deliberate part of the research. For example, in Section 7.4 we listed criteria that limits the applicability of this work to particular types of organisations or organisational processes. Other limitations though became apparent during the progress of the research. While we certainly cannot provide a complete list of all of the many things that the work is unable to address we can present a list of some of the more interesting limitations.

Structuring extension modules One of the limitations of this work has to do with the lack of information we provide on how to build and structure extension modules. For the most part, we simply describe an approach in which various elements (e.g. relations, functions, classes) can be placed in a file and then referenced by other files. In practice though, more research is required, for example, to understand how these dependencies are managed, how conflicts in referenced extensions are resolved (e.g. if "order" is defined in two different ways for two separate extensions), and the role that versioning of extensions could play.

Generic translation While we advocate a translation approach to process knowledge sharing in this thesis, our set of implemented translation modules are limited in their usefulness. The biggest limitation has to do with what they say about a generic approach to translation and concept mapping. Uschold et al. describes a need to understand the basic, generic steps inside such translation modules [Ushold *et al.*, 1998] to help enable ontological reuse. This information would assist translation authors by contributing toward a repeatable, well-defined method for constructing translators. This area is currently a very active area of research.

Ontologies of actions and objects In describing domain ontologies, Gómez–Pérez identifies the differences between object and activity ontologies

[Gómez-Pérez, 1998]. Our work provides partial support for structuring the acquisition of activity ontologies by applying an approach from requirements engineering. We haven't provided adequate support for the necessary and orthogonal acquisition of object ontologies. While we do support the *expression* of activity relatable objects, it isn't intended that the knowledge of how these objects are engineered is to be fixed within CPF. We do feel though that there needs to be a method that unites the acquisition of both collections of elements in a domain. Some approaches, such as Methonology [Gómez-Pérez *et al.*, 1996, Fernández *et al.*, 1997], provide examples of methods that could be used to help expand the requirements analysis CPF phase to provide a more comprehensive ontological engineering method.

Flexibility of expression While we have emphasised the strength of a highly flexible approach to constraint expression (see Section 3.3.2.12), but we have also discussed the fact that this involves a trade-off in efficiency (Section 2.7.2). Many elements and concepts can be expressed in this interlingua, but it is assumed that a translation step is required to map this knowledge into a more efficient format for various reasoning purposes. This mapping requires a "per constraint type" understanding of the grammar. While other approaches may propose interlinguas that can be reasoned over more efficiently they each incur their own limitations in expressiveness.

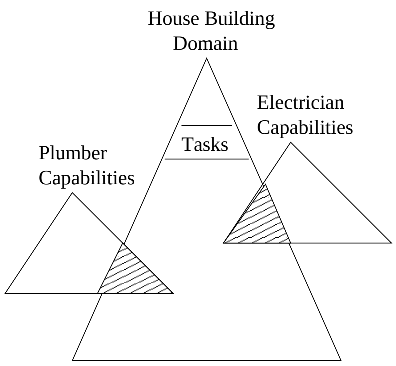

Figure 7.2: Overlapping models of a domain and agent capabilities

Eliciting agent capabilities The CPF approach towards eliciting knowledge to be used to synthesise new configurations of organisational processes is very closely tied to a particular domain task and doesn't adequately address agent capability models. For example, consider the house building domain which we illustrate in Figure 7.2. Our knowledge of this domain intersects with the knowledge of some capabilities of the plumber and electrician agents. For example, an electrician may be capable of doing many things such as installing wiring and also repairing or replacing old wiring. While in CPM we might model the electrician using a bounding viewpoint, the only capability we will probably choose to describe is the install wiring action. We believe that a generic method which captures agent capabilities and then uses parts of these capabilities to construct a domain specification would enable a more hygienic management approach.

Tracking changes to requirements, rationale and constructs CPF provides the ability to express requirements using CPM and the capability to translate them to actual constructs in the domain specification. While this is certainly a helpful first step towards encouraging a more well-defined engineering approach there is much work that remains to be done regarding the connection between requirements and designed constructs. Ideally, a mechanism for maintaining a link between the two would help coordinate changes on either side of the equation allowing updates, or at least notification of required updates.

CPF also provides the ability to express design rationale using the design space analysis approach which we adopted. Again, we believe that this adds value to a principled methodology of managing process knowledge, but work remains to make this knowledge a more effective part of the framework. Similar to the requirements, rationale needs to be linked in some way to the process constructs. In fact, such research is currently underway, for example, with the ProcessLink system which utilises and extends a general model of design change propagation (Redux) that makes design rationale active by tracking several aspects of a plan's validity and informing agents when it changes [Petrie *et al.*, 1999].

Mixed-Initiative approach CPF provides a very coarse-grained notion of mixed-initiative planning or process design interaction between people and systems. An entire linear representation (i.e. a CPL file) of some process specification can be accessed and sent across the internet using the CPF toolset communication capabilities (i.e. FTP get/put). These specifications can be translated into appropriate languages as needed to interface with systems. The state of incomplete plans or process designs can embedded by using issue constraints. A more realistic, robust approach might be required to include interaction with defined interfaces (e.g. plan server interface, plan editor interface, etc.) and knowledge of some kind of messaging service (e.g. the KQML work we mentioned in Section 6.2.5).

Mapping requirements While we have been able to show that the mappings between a requirements engineering approach and an initial domain specification for AI planning is possible, we believe that more work can be done to improve this mapping. For example, in CPM we identify data flows as being various types: data containing information, event data, and control data. In our current implementation we have pushed a lot of the work in interpreting this information into the detailed domain development phase, favouring a more straight-forward CPM to CPD translation. It is possible that work looking into patterns of requirements might produce more automated translation capabilities.

7.6 Further Research

In this section we consider the future of this work and the need for further research. Some of this research might be aimed at the replacement or modification of existing CPF components or the introduction of improvements to the design of the CPF architecture.

In this work we have taken a novel stride toward an understanding of how we can apply research from AI planning to the management of organisational processes. In doing so we have built this work on top of existing research from a range of areas. Looking ahead a bit we can envision ways that this contribution to knowledge may also be used as a building block for further research.

The first item to point out in considering further research that either builds on or draws from this thesis is the modularity of the framework. Most of these modules exist behind translators which decouple the framework components. This means that new components could be snapped into the framework which, for example, address some of the limitations we discussed in Section 7.5. One example would be to replace the CORE methodology component (in the requirements analysis phase) with a knowledge acquisition approach that adequately addresses the requirements of elicitation of domain knowledge in multi-user environments (cf. Internet-based Multi-agent Problem Solving (IMPS) architecture [Crow and Shadbolt, 1999]).

We discussed the need to further explore and understand the structure and role of translation as a facilitator of knowledge sharing, especially in the face of "living ontologies" which permit aggregations of ontological extensions to a process core. Given this understanding, partially automated generation of such translation modules could help reduce the time required to build these links.

Another avenue of research could look into integrating both this translation model with an interface model of knowledge sharing. For example, the interaction between the domain editor, AI planning system, and the process editor (which displays current process designs) would benefit by being able to establish a more "conversational" interaction. One way in which this could be done is by opening up and exposing the capabilities or interfaces of the tools using techniques from distributed computing. This might utilise standards or technologies such as CORBA, Enterprise JavaBeans, RMI, or Jini. The Java Native Interface (JNI) could be used to help establish links with existing planning systems which could then be wrapped inside of server stub implementations. Such implementations will certainly be required for true mixed-initiative approaches.

Other examples of further research which we have mentioned include connecting requirements and design rationale to domain specifications which could perhaps be done through the use of reason maintenance systems (RMS). We might also consider research which can broaden the applicability of this approach beyond the type of organisations described in Section 7.4. For example, looking at ways that this approach can be adapted for domains that have dynamic, continuous processes.

7.7 Conclusion

The problem we have addressed in this dissertation is that of designing a pragmatic framework for integrating the synthesis and management of organisational process knowledge which is based on domain-independent AI planning and plan representations. We defined a lifecycle of this knowledge which begins with a methodological approach to acquiring information about the process domain. We showed that this initial domain specification can be translated into a common constraint-based model of activity which can then be operationalised for use in an AI planner. This model of activity is ontologically underpinned and can be expressed with a flexible and extensible language based on a sorted first-order logic. Our approach shows how synthesised or modified processes/plans can be translated to and from the common representation in order to support knowledge sharing, visualisation and mixed-initiative interaction. This research addressed the following objectives:

Objective 1: Investigate the application of a requirements engineering methodology for eliciting process domain information.

Objective 2: Examine the advantages of a single core representation that can be extended as appropriate for various tasks. This will build on existing ARPI-sponsored research involving the <I-N-OVA> constraint model of activity.

Objective 3: Define a set of tools required to manage both process domain, and individual plan/process information (e.g. to support visualisation, communication, translation)

Objective 4: Explore some of the extensions required and issues involved in applied scenarios from business, manufacturing and military scenarios.

7.7.1 Accomplishments

Our solution focused on a set of framework components which provide methods, tools and representations to accomplish the objectives stated in Section 7.7. This work united past and present Edinburgh research on planning and infused it with perspectives from design rationale, requirements engineering, and process knowledge sharing.

One of the early accomplishments of this research involved a review of rationale in planning [Polyak and Tate, 1998]. This led to our perspective of decision rationale which could be expressed and communicated with notations developed in design rationale. We recognised that the formulation of this decision knowledge could be viewed and communicated as an extension to some core set of shared process entities.

In looking at shared process languages, we evaluated a number of ARPI-funded, shared AI planning and scheduling representations against a set of requirements developed for NIST's Process Specification Language (PSL). Based on this work, we found that <I-N-OVA> provided the most appropriate flexible model with which to anchor our shared representation [Polyak and Tate, 1997].

Following the lead from others in this work we defined a sorted first-order logic with which to express this constraint-based process knowledge in a very flexible way [Polyak and Tate, 1999, Polyak, 1998b]. We envisioned extensions to the core ontology to be added (e.g. decision rationale) much in the same way as was currently adopted in the PSL and PIF work.

In parallel with this, we developed a set of scenarios for use in both this thesis work and to be used in process standards work for related projects [Polyak, 1998e, Polyak, 1998f, Polyak, 1997a, Polyak and Aitken, 1998]. These scenario descriptions were distributed to project members and were used to focus work in this area.

We went back to earlier O-Plan research ideas on adapting the CORE methodology for engineering process domains [Wilson, 1984]. We extended this work and integrated it with the emerging shared process knowledge approach. The aim was to arm domain specialists and experts with a principled set of methods and intermediate models.

We further updated O-Plan work represented by the Task Formalism workstation [Tate and Currie, 1984, Tate and Currie, 1985] to support process visualisation, editing, and communication. This work addressed issues of translation to and from the shared process representation. This introduced, for example, a specification step which proceeds the development of an operational AI planning domain. We also explored support for the integration of tools that can provide "expert system" analyses, as envisioned in the TF workstation design. In doing so, we looked into temporal mapping issues between timepoint and interval theories [Polyak, 1998c].

The accomplishments of this thesis, combined with the further research which we have mentioned in Section 7.6, aid in efforts aimed at closing the gap between the theoretically clean and applied research. The understanding derived from this work builds on a foundation of research into improved methods of synthesising and managing organisational process knowledge.

Bibliography

[Aarup *et al.*, 1994] M. Aarup, M.M. Arentoft, Y. Parrod, J. Stader, I. Stokes, and H. Vadon. OPTIMUM-AIV: A knowledge based planning and scheduling system for spacecraft AIV. In M. Fox and M. Zweben, editors, *Knowledge Based Scheduling*. Morgan Kaufman, 1994.

[Agosta, 1994] J.M. Agosta. Constraining influence diagram structure by generative planning: An application to the optimization of oil spill response. In *Proceedings of the 12th Conference on Uncertainty in AI*, pages 11–19. AAAI Press, 1994.

[Aitken and Tate, 1997] S. Aitken and A. Tate. Process modelling of the TIE 97-1 demonstration: Modelling complex technologies using ACP terminology. Artificial Intelligence Applications Institute ISAT-AIAI-TR-6, University of Edinburgh, Edinburgh, Scotland, 1997.

[Albus *et al.*, 1989] J.S. Albus, H.G. McCain, and R. Lumia. NASA/NBS standard reference model for telerobot control system architecture (NASREM). Technical Report Tech. Note 1235, National Bureau of Standards, 1989.

[Alderman, 1997] N. Alderman. Business process analysis: A system-wide perspective. In *Proceedings of the Business Process Track at the British Academy of Management Conference*, London, UK, September 1997. ESRC Business Processes Resource Centre.

[Allen and Koomen, 1983] J. Allen and J. Koomen. Planning using a temporal world model. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-83)*, 1983.

[Allen *et al.*, 1990] J.F. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning*. Morgan Kaufmann Publishing Co., Palo Alto, CA, 1990.

[Allen *et al.*, 1996] J.F. Allen, G.M. Ferguson, and L.K. Schubert. Planning in complex worlds via mixed-initiative interaction. In Tate [1996a], pages 53–60.

[Allen, 1984a] J. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1984.

[Allen, 1984b] J.F. Allen. Towards a general theory of action and time. In Allen et al. [1990], pages 464–479.

[Anderson *et al.*, 1998] C.R. Anderson, D.E. Smith, and D.S. Weld. Conditional effects in graphplan. In Simmons et al. [1998], pages 44–53.

[Arentoft *et al.*, 1991] M.M. Arentoft, Y. Parrod, J. Stader, I. Stokes, and H. Vadon. OPTIMUM-AIV: A planning and scheduling system for spacecraft AIV. *Telematics and Informatics*, 8(4):239–252, 1991.

[Arntzen *et al.*, 1995] B.C. Arntzen, G.G. Brown, T.P. Harrison, and L. Trafton. Global supply chain management at digital equipment corporation. *Interfaces*, Jan.–Feb., 1995.

[Aylett and Jones, 1996] R. Aylett and S. Jones. Planner and domain: Domain configuration for a task planner. *International Journal of Expert Systems*, 9(2):279–318, 1996.

[Aylett *et al.*, 1997] R. Aylett, G. Petley, P.W. Chung, J. Soutter, and A. Rushton. Planning and chemical plant operating procedure synthesis: a case study. In *Proceedings of the 4th European Conference on Planning*, 1997.

[Aylett *et al.*, 1998] R. Aylett, J. Soutter, G. Petley, P.W. Chung, and A. Rushton. AI planning in a chemical plant domain. In *Proceedings of the European Conference on Artificial Intelligence*, 1998.

[Bañares–Alcántara, 1991] R. Bañares–Alcántara. Representing the engineering design process: Two hypothesis. *Computer-aided Design*, 23:595–604, 1991.

[Baecker, 1993] Ronald M. Baecker, editor. *Readings in Groupware and Computer Supported Collaborative Work*. Morgan Kaufmann, San Mateo, CA, 1993.

[Ballinger *et al.*, 1993] G. Ballinger, R. Bañares–Alcántara, and J. King. Using an IBIS to record design rationale. Department of Chemical Engineering ECOSSE Technical Report 1993-17, University of Edinburgh, Edinburgh, Scotland, 1993.

[Batchu *et al.*, 1995] S. Batchu, S. Kambhampati, H. Kartheek, and J. Shah. An iterative and interactive approach for process planning. Technical Report ASU CSE TR 95-023, Arizona State University, Tempe, AZ, USA, 1995.

[Beck, 1993] H. Beck. The management of job-shop scheduling constraints. Artificial Intelligence Applications Institute AIAI Technical Report No 118, University of Edinburgh, Edinburgh, Scotland, 1993.

[Bell *et al.*, 1986a] C.E. Bell, K.W. Currie, and A. Tate. Managing scheduling and resource usage constraints in O-Plan. Technical Report 4, Alvey Planning SIG, Sunningdale, UK, 1986.

[Bell *et al.*, 1986b] C.E. Bell, K.W. Currie, and A. Tate. Using temporal constraints to restrict search in a planner. Technical Report AIAI-TR-5, Artificial Intelligence Applications Institute (AIAI), Edinburgh, Scotland, 1986.

[Benjamins *et al.*, 1996] V.R. Benjamins, L. Nunes de Barros, and A. Valente. Constructing planners through problem-solving methods. In *Proceedings of KAW '96 Banff*, pages 14.1–14.20, 1996.

[Bernaras *et al.*, 1996] A. Bernaras, I. Laresgoiti, and J. Corera. Building and reusing ontologies for electrical network applications. In *Proceedings of the European Conference on Artifical Intelligence (ECAI) '96*, pages 298–302, 1996.

[Bienkowski, 1996] M.A. Bienkowski. SOCAP: System for operations crisis action planning. In Tate [1996a], pages 70–76.

[Bisiani *et al.*, 1988] R. Bisiani, F. Lecouat, and V. Ambriola. A planner for the automation of programming environment tasks. In *21st International Hawaii Conference on System Sciences*, January 1988.

[Blum and Furst, 1995] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, 1995.

[Blythe *et al.*, 1996] J. Blythe, M. Veloso, and L.E. de Souza. The prodigy user interface. Department of Computer Science Technical Report, Carnegie Mellon University, Pittsburgh, PA, 1996.

[Booch *et al.*, 1998] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[Borgida *et al.*, 1985] A. Borgida, S. Greenspan, and J. Mylopoulos. Knowledge representation as the basis for requirements specification. *IEEE Computer*, pages 82–90, April 1985.

[BPRAG, 1995] BPRAG. Designing tools to support business process reengineering. Business Process Re-engineering Advisory Group Report July, Enterprise Integration Laboratory, Department of Industrial Engineering, University of Toronto, Toronto, Ontario M5S 1A4, 1995.

[Brachman and Levesque, 1985] R.J. Brachman and H.J. Levesque. *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, 1985.

[Bratman *et al.*, 1988] M.E. Bratman, D.J. Israel, and M.E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[Bratman, 1987] M.E. Bratman. *Intention, Plans and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[Breuker and van de Velde, 1994] J. Breuker and W. van de Velde. *The CommonKADS Library for Expertise Modelling: reusable components for artificial problem solving*. IOS Press, Amsterdam, Tokyo, 1994.

[Breuker, 1994] J. Breuker. A suite of problem types. In breuker94a [1994], pages 57–87.

[Buckingham Shum *et al.*, 1997] S. Buckingham Shum, A. MacLean, V. Bellotti, and N. Hammond. Graphical argumentation and design cognition. *Human-Computer Interaction*, 12(3):267–300, 1997.

[Buckingham Shum, 1991a] S. Buckingham Shum. Cognitive dimensions of design rationale. In D. Diaper and N.V. Hammond, editors, *People and Computers VI*, pages 1–13, Cambridge, 1991. Cambridge University Press.

[Buckingham Shum, 1991b] S. Buckingham Shum. Cognitive issues in representing design reasoning as hypertext. *SIGCHI Bulletin*, 23(1):38–40, 1991.

[Burstein and McDermott, 1994] M.H. Burstein and D. McDermott. Mixed-initiative military planning: Directions for future research and development. In Burstein [1994], pages 467–483.

[Burstein, 1994] M.H. Burstein, editor. *Proceedings of the ARPA/Rome Laboratory Knowledge-based planning and Scheduling Initiative Workshop*, Tuscon, AZ, 1994. Morgan Kaufmann.

[Calvanese *et al.*, 1998] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, pages 2–13, 1998.

[Carbonell *et al.*, 1990] J.G. Carbonell, C.A. Knoblock, and S. Minton. Prodigy: An integrated architecture for planning and learning. In K. VanLehn, editor, *Architectures for Intelligence*, Hinsdale, NJ, 1990. Erlbaum.

[Caron *et al.*, 1994] J.R. Caron, S.L. Jarvenpa, and D.B. Stoddard. Business reengineering at CIGNA corporation: Experiences and lessons learned from the first five years. *Management Information Systems Quarterly*, 18(3), 1994.

[Cartin, 1993] T.J. Cartin. *Principles and Practices of TQM.* American Society for Quality, New York, NY, USA, 1993.

[Chan Kim and Mauborgne, 1997] W. Chan Kim and R. Mauborgne. Fair process: Managing in the knowledge economy. *Harvard Business Review*, July–August, 1997.

[Chan, 1995] F.Y. Chan. *The Round Trip Problem: A Solution for the Process Handbook.* M.S. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (MIT), 1995.

[Chandrasekaran, 1987] B. Chandrasekaran. Towards a funtional architecture for intelligence based on generic information processing tasks. In *Proceedings of the Tenth IJCAI*, pages 1183–1192, 1987.

[Chang and Lee, 1973] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science and Applied Mathematics Series. Academic Press, New York, NY, 1973.

[Chapman, 1987] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[Charniak and McDermott, 1985] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Co., 1985.

[Chen-Burger and Robertson, 1998] J. Chen-Burger and D. Robertson. Formal support for an informal business modelling method. In *10th International Conference on Software Engineering and Knowledge Engineering (SEKE'98)*, San Francisco, USA, 1998.

[Chien and Mortenson, 1996] S. Chien and H. Mortenson. The multimission VICAR planner: A fielded planning system for scientific data analysis. In Ghallab and Milani [1996], pages 371–382.

[Chien *et al.*, 1996] S. Chien, R. Hill, X. Wang, T. Estlin, K. Fayyad, and H. Mortenson. Why real-world planning is difficult: A tale of two applications. In Ghallab and Milani [1996], pages 287–298.

[Chien, 1996] S.A. Chien. Intelligent tools for planning knowledge base development and verification. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Advances in Knowledge Acquisition, 9th European Knowledge Acquisition Workshop* EKAW'96, pages 321–337. Springer, 1996.

[Chung and Goodwin, 1994] P.W.H. Chung and R. Goodwin. Representing design history. In J.S. Gero and F. Sudweeks, editors, *Artificial Intelligence in Design '94*, pages 735–752, Netherlands, 1994. Kluwer Academic Publishers.

[Clocksin and Mellish, 1981] W.F. Clocksin and C.S. Mellish. *Programming in Prolog.* Springer Verlag, 2nd edition, 1981.

[Cohen, 1995] Paul R. Cohen. *Empirical Methods for Artificial Intelligence.* MIT Press, Cambridge, Massachusetts. London, England, 1995.

[Cohn, 1985] A.G. Cohn. On the solution of schubert's steamroller in many sorted logic. In *IJCAI*, pages 1169–1174, 1985.

[Collins and Pryor, 1993] G. Collins and L. Pryor. On the misuse of filter conditions: a criticial analysis. In C. Backstrom and E. Sandewall, editors, *Current Trends in AI Planning.* IOS Press, Amsterdam, Netherlands, 1993.

[Conklin and Begeman, 1988] E.J. Conklin and M. L. Begeman. gIBIS: A hypertext tool for explanatory policy discussion. *ACM Transactions on Office Information Systems*, 6:303–331, 1988.

[Conklin and Yakemovic, 1991] E.J. Conklin and KC.B. Yakemovic. A process-oriented approach to DR. *Human Comp. Inter.*, 6:357–391, 1991.

[Cottam and Shadbolt, 1996] H. Cottam and N. Shadbolt. Domain and system influences in problem solving models for planning. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Advances in Knowledge Acquisition, 9th European Knowledge Acquisition Workshop* EKAW'96. Springer, 1996.

[Cottam *et al.*, 1995] H. Cottam, N. Shadbolt, J. Kingston, H. Beck, and A. Tate. Knowledge level planning in the search and rescue domain. In *Research and Development in Expert Systems XII, proceedings of BCS Expert Systems'95*, Cambridge, 11–13 December, 1995.

[Cottam *et al.*, 1998] H. Cottam, N. Shadbolt, and N. Milton. The use of ontologies in a decision support system for business process re-engineering. In *Proceedings of IT and KNOWS Conference of the 15th IFIP World Computer Congress*, Vienna/Budapest, 1998.

[Crow and Shadbolt, 1998] L.R. Crow and N.R. Shadbolt. Internet agents for knowledge engineering. In *Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1998. SRDG Publications.

[Crow and Shadbolt, 1999] L.R. Crow and N.R. Shadbolt. Knowledge engineering with software agents. In *Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace*, Stanford, CA, 1999.

[Currie and Tate, 1991] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

[Curtis *et al.*, 1992] B. Curtis, M.I. Kellner, and J. Over. Process modeling. *Communications of the ACM*, 35(9):75–90, 1992.

[Curwen, 1991] P. Curwen. System development using the CORE method. Military Aircraft Ltd. BAe/WIT/ML/GEN/SWE/1227, British Aerospace, PLC, Warton Aerodrome, Preston, UK, 1991.

[Daniel, 1983] L. Daniel. Planning and operations research. In *Artificial Intelligence: Tools, Techniques and Applications.* Harper and Row, New York, 1983.

[Davenport and Prusak, 1998] T.H. Davenport and L. Prusak. *Working Knowledge: How Organizations Manage What They Know.* Havard Business School Press, Boston, MA, USA, 1998.

[Davenport *et al.*, 1996] T.H. Davenport, S.L. Jarvenpaa, and M.C. Beers. Improving knowledge work process. *Sloan Management Review*, pages 53–65, 1996.

[Davenport, 1993] T.H. Davenport. *Process Innovation Re-engineering Work through Information Technology.* Havard Business School, Boston, MA, USA, 1993.

[Davidow and Malone, 1992] W.H. Davidow and M.S. Malone. *The Virtual Corporation.* Harper Collins Business, New York, NY, USA, 1992.

[Davis and Bonnell, 1991] J.P. Davis and R.D. Bonnell. A framework for constructing visual knowledge specifications in acquiring organizational knowledge. *Knowledge Acquisition*, 3(1):79–115, 1991.

[Davis and Botkin, 1994] S. Davis and J. Botkin. The coming of knowledge-based business. *Harvard Business Review*, September-October:165–170, 1994.

[Davis, 1990] E. Davis. *Representations of Commonsense Knowledge.* Morgan Kaufmann Publishers, Inc., 1990.

[Davis, 1993] A.M. Davis. *Software Requirements: Objects, Functions, and States.* Prentice Hall Publishers, 1993.

[de Kleer and Williams, 1986] J. de Kleer and B. Williams. Reasoning about multiple faults. In *Proceedings of The 5th National Conference on Artificial Intelligence*, pages 132–139, 1986.

[de Kleer, 1986] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[Dean and Kambhampati, 1995] T. Dean and S. Kambhampati. Planning and scheduling. In A. Tucker, editor, *CRC Handbook of Computer Science and Engineering*. CRC Press Inc., 1995.

[Dean *et al.*, 1993] T. Dean, L.P. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 574–579, 1993.

[Decker *et al.*, 1997] Keith Decker, Katia Sycara, and Mike Williamson. Middle-agents for the internet. In *Proc. 15th IJCAI*, pages 578–583, Nagoya, Japan, August 1997. Morgan Kaufmann.

[DeKleer and Williams, 1987] J. DeKleer and B. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[desJardins, 1996] M. desJardins. Knowledge acquisition tools for planning systems. In Tate [1996a], pages 53–61.

[Dobson *et al.*, 1994] J.E. Dobson, A.J.C. Blyth, J. Chudge, and R. Strens. The ORDIT approach to organisational requirements. In M. Jirotka and J. Goguen, editors, *Requirements Engineering: Social and Technical Issues*, pages 87–106. Academic Press, London, 1994.

[Domingue *et al.*, 1993] J. Domingue, E. Motta, and S. Watt. The emerging VITAL workbench. In *Proceedings of the 7th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW '93)*, Toulouse, France, 1993.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(2):231–272, 1979.

[Doyle, 1994] J. Doyle. A reasoning economy for planning and replanning. In Burstein [1994], pages 35–43.

[Doyle, 1996] J. Doyle. Toward rational planning and replanning rational reason maintenance, reasoning economies, and qualitative preferences. In Tate [1996a], pages 130–135.

[Drabble and Beck, 1995] B. Drabble and H. Beck. Intelligent systems for business planning. Artificial Intelligence Applications Institute, AIAI AIAI-TR-178, University of Edinburgh, Edinburgh, Scotland, 1995.

[Drabble and Tate, 1994] B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In *Proceedings of the Second International Conference on Planning Systems, Chicago, June*. AAAI Press, 1994.

[Drabble *et al.*, 1997a] B. Drabble, J. Dalton, and A. Tate. Repairing plans on–the–fly. In *Proceedings of the NASA Workshop on Planning and Scheduling for Space*, Oxnard CA, USA, 1997.

[Drabble *et al.*, 1997b] B. Drabble, T. Lydiard, and A. Tate. Process steps, process products and system capabilities. Technical Report ISAT-AIAI/TR/4 Version 2, Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, Scotland, April 1997.

[Drabble, 1990] B. Drabble. Mission scheduling for spacecraft: Diaries of T-Sched. In *Expert Planning Systems*, pages 76–81. Institute of Electrical Engineers, 1990.

[Drabble, 1995] B. Drabble. Artificial intelligence for project planning. Artificial Intelligence Applications Institute AIAI-TR-206, University of Edinburgh, Edinburgh, Scotland, 1995.

[Drabble, 1996] B. Drabble, editor. *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, Edinburgh, Scotland, May 1996. Morgan Kaufmann.

[Draper *et al.*, 1994] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, June 1994.

[Drummond and Tate, 1992] M.E. Drummond and A. Tate. PLANIT interactive planner's assistant – rationale and future directions. Artificial Intelligence Applications Institute AIAI-TR-108, University of Edinburgh, 1992.

[Easterbrook and Nuseibeh, 1996] S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *Soft. Engin. Journ.*, January, 1996.

[Eisenstadt *et al.*, 1990] M. Eisenstadt, J. Domingue, T. Rajan, and E. Motta. Visual knowledge engineering. *IEEE Transactions on Software Engineering*, 16(10):1164–1177, 1990.

[Engelmore and Morgan, 1988] Robert Engelmore and Tony Morgan, editors. *Blackboard Systems.* Addison-Wesley, Reading, MA, 1988.

[Eriksson *et al.*, 1995] Henrik Eriksson, Yuval Shahar, Samson W. Tu, Angel R. Puerta, and Mark A. Musen. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326, December 1995.

[Ernst, 1997] J. Ernst. Introduction to CDIF. Technical Report EIA–PNCDIF, CASE Data Interchange Format Division, Electronic Industries Association, 1997.

[Erol, 1995] K. Erol. *Hierarchical Task Network Planning: Formalisation, Analysis, and Implementation.* Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park, USA, 1995.

[Eshghi, 1988] K. Eshghi. Abductive planning with event calculus. In *Proceedings of the 5th International Conference on Logic Programming*, pages 562–579, 1988.

[Etzioni *et al.*, 1993] Oren Etzioni, Henry M. Levy, Richard B. Segal, and Chandramohan A. Thekkath. OS agents: Using AI techniques in the operating system environment. Technical Report 93-04-04, University of Washington, Seattle, WA, April 1993.

[Etzioni, 1997] Oren Etzioni. Moving up the information food chain. *AI Magazine*, 18(2):11–18, Summer 1997.

[Ferguson and Allen, 1994] G. Ferguson and J. Allen. Arguing about plans: Plan representation and reasoning for mixed-initiative planning. In Burstein [1994], pages 123–131.

[Fernández *et al.*, 1997] M. Fernández, A. Gómez-Pérez, and N. Juristo. Methontology: From ontological art towards ontological engineering. In *Workshop on Ontological Engineering, Spring Symposium Series, AAAI97*, Stanford, USA, 1997.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes, 1982] R. Fikes. A commitment-based framework for describing informal cooperative work. *Cognitive Science*, 6:331–347, 1982.

[Finin, 1992] T. Finin. Specification of the KQML agent communication language. Technical Report EIT TR 92-04, Enterprise Integration Technologies, Palo Alto, CA, 1992.

[Finkelstein *et al.*, 1994] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. *Trans Software Eng*, 20(8):569–578, 1994.

[Flanagan, 1997] D. Flanagan. *Java in a Nutshell, 2nd Edition.* O'Reilly and Associates, Inc., 1997.

[Fowler *et al.*, 1996] N. Fowler, T.D. Garvey, S.E. Cross, and M. Hoffman. Overview: ARPA-Rome laboratory knowledge-based planning and and scheduling initiative (ARPI). In Tate [1996a], pages 3–9.

[Fox and Long, 1998] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research (JAIR)*, 9, 1998.

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. Department of Computer Science, Technical Report 1/99, University of Durham, 1999.

[Fox *et al.*, 1993] M.S. Fox, J.F. Chionglo, and F.G. Fadel. A common-sense model of the enterprise. In *Proceedings of the Second Industrial Engineers Research Conference (IERC)*, Norcross, GA, USA, 1993. Institute for Industrial Engineers.

[Fraser and Tate, 1995] J. Fraser and A. Tate. The enterprise tool set – an open enterprise architecture. In *Proceedings of the Workshop on Intelligent Manufacturing Systems, International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

[Fuchs *et al.*, 1990] J.J. Fuchs, A. Gasquet, B. Olalainty, and K.W. Currie. PlanERS-1: An expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems*, pages 70–75, Brighton, United Kingdom, 1990. Institute of Electrical Engineers.

[Fukunaga *et al.*, 1997] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan. Towards and application framework for automated planning and scheduling. In *Proceedings of IEEE Aerospace Conference*, 1997.

[Gallier, 1986] Jean H. Gallier. *Logic for Computer Science*. Harper and Row, New York, NY, 1986.

[Gattorna and Walters, 1996] J.L. Gattorna and D.W. Walters. *Managing the Supply Chain*. Macmillan Press Ltd., London, UK, 1996.

[Gelfond *et al.*, 1991] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In *AAAI Spring Symposium Series on Logical Formilizations of Commonsense Reasoning*, pages 59–69, 1991.

[Genesereth and Nilsson, 1988] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., 1988.

[Genesereth *et al.*, 1992] Michael R. Genesereth, Richard E. Fikes, Daniel Bobrow, Ronald Brachman, Thomas Gruber, Patrick Hayes, Reed Letsinger, Vladimir Lifschitz, Robert MacGregor, John McCarthy, Peter Norvig, Ramesh Patil, and Len Schubert. Knowledge interchange format version 3.0 reference manual. Report Logic-92-1, Stanford University, Stanford, CA, June 1992.

[Genesereth, 1991] Michael R. Genesereth. Knowledge interchange format. In *Proc. 2nd KR*, pages 599–600, Cambridge, MA, 1991. Morgan Kaufmann.

[Georgeff and Ingrand, 1989] M.P. Georgeff and F.F. Ingrand. Decision-making in an embeded reasoning system. In *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, Menlo Park, CA, 1989. AAAI Press.

[Georgeff *et al.*, 1989] M.P. Georgeff, F.F. Ingrand, and A.L. Lansky. Procedural reasoning system. Technical Report User Guide, SRI International Artificial Intelligence Center, Menlo Park, CA, 1989.

[Georgeff, 1987] M.P. Georgeff. Planning. *Ann. Rev. Comput. Sci.*, 2:359–400, 1987.

[Ghallab and Milani, 1996] M. Ghallab and A. Milani, editors. *New Directions in AI Planning*. IOS Press, 1996.

[Ghosh *et al.*, 1992] S. Ghosh, J. Hendler, S. Kambhampati, and B. Kettler. UM Nonlin Version 1.2.2 User Manual. Department of Computer Science, University of Maryland, College Park, MD, 1992.

[Giarratano and Riley, 1994] J.C. Giarratano and C. Riley. *Expert Systems: Principles and Programming*. PWS Pub. Co., Boston, MA., 1994.

[Giarratano, 1994] J.C. Giarratano. CLIPS 6.0 user's guide. Software Technology Branch JSC-25013, Lyndon B. Johnson Space Center, Information Systems Directorate, 1994.

[Gil *et al.*, 1995] Y. Gil, M. Veloso, S. Chien, D. McDermott, and D. Nau. Symposium preface. In *Planning and Learning: On to Real Applications. Papers from the 1994 AAAI Fall Symposium, number FS-94-01.* AAAI Press, American Association for Artificial Intelligence, ISBN 0-929280-75-X, 1995.

[Gil, 1992] Y. Gil. *Acquiring Domain Knowledge for Planning By Experimentation.* Ph.D. Thesis, Carnegie-Mellon University, 1992.

[Ginsberg, 1993] M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[Goldman and Boddy, 1994] R. Goldman and M. Boddy. Epsilon–safe planning. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 253–261, 1994.

[Gómez-Pérez *et al.*, 1996] A. Gómez-Pérez, M. Fernández, and A. De Vicente. Towards a method to conceptualize domain ontologies. In *Workshop on Ontological Engineering, ECAI'96*, pages 41–51, 1996.

[Gómez-Pérez, 1998] A. Gómez-Pérez. Knowledge sharing and reuse. In Liebowitz, editor, *The handbook on Applied Expert Systems.* CRC Press, 1998.

[Gordon *et al.*, 1993a] T.F. Gordon, J. Hertzberg, and A. Horz. A planner for beautifying business graphics. Technical Report TASSO-Report 45, BMFT-Verbundprojekt TASSO, 1993.

[Gordon *et al.*, 1993b] T.F. Gordon, J. Hertzberg, and A. Horz. The qwertz toolbox reference manual. Technical Report arbeitspapier 835, GMD, 1993.

[Gray *et al.*, 1997] P.M.D. Gray, A. Preece, N.J. Fiddian, W.A. Gray, T.J.M. Bench-Capon, M.J.R. Shave, N. Azarmi, M. Wiegand, M. Ashwell, M. Beer, Z. Cui, B. Diaz, S.M. Embury, K. Hui, A.C. Jones, D.M. Jones, G.J.L. Kemp, E.W. Lawson, K. Lunn, P. Marti, J. Shao, and P.R.S. Visser. KRAFT: Knowledge fusion from distributed databases and knowledge bases. In *Database and Expert System Applications (DEXA' 97)*, 1997.

[Green, 1969] C. Green. Application of theorem proving to problem solving. In Allen et al. [1990], pages 67–87.

[Greenspan *et al.*, 1982] S.J. Greenspan, J. Mylopoulos, and A. Borgida. Capturing more world knowledge in the requirements specification. In *Proceedings of the Sixth International Conference on Software Engineering*, Los Alamitos, CA, 1982. IEEE–CS Press.

[Greenspan, 1984] S.J. Greenspan. *Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition.* Ph.D. Thesis, Department of Computer Science, University of Toronto, 1984.

[Gross *et al.*, 1993] D. Gross, J. Allen, and R.D. Traum. The TRAINS-91 dialogues. Department of Computer Science TRAINS Technical Note 92-1, University of Rochester, Rochester, NY, 1993.

[Gruber, 1993] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gruninger and Pinto, 1995] M. Gruninger and J. Pinto. A theory of complex actions for enterprise modelling. In *AAAI Spring Symposium Series on Extending Theories of Action: Formal Theory and Practical Applications*, 1995.

[Gruninger, 1996] M. Gruninger. Characterization of tools for business process reengineering. Business Process Re-engineering Advisory Group Report July, Enterprise Integration Laboratory, Department of Industrial Engineering, University of Toronto, Toronto, Ontario M5S 1A4, 1996.

[Guha and Lenat, 1990] R. V. Guha and Douglas B. Lenat. Cyc: A midterm report. *AI Magazine*, 11(3):32–59, Fall 1990.

[Guha and Lenat, 1994] R. V. Guha and Douglas B. Lenat. Enabling agents to work together. *Communications of the ACM*, 37(7):127–142, July 1994.

[Guida and Tasso, 1994] G. Guida and C. Tasso. *Development of Knowledge-Based Systems: From Life Cycle to Development Methodology.* John Wiley and Sons, Chichester, UK, 1994.

[Gupta *et al.*, 1998] S.K. Gupta, D.S. Nau, and W.C. Regli. IMACS: A case study in real-world planning. *IEEE Expert and Intelligent Systems*, 13(3), May/June 1998.

[Haigh and Veloso, 1998] K.Z. Haigh and M.M. Veloso. Planning, execution and learning in a robotic agent. In Simmons et al. [1998], pages 120–127.

[Hammer and Champy, 1994] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* Nicholas Brealey Publishing, London, 1994.

[Hammer, 1996] M. Hammer. *Beyond Reengineering.* Harper Collins Business, London, UK, 1996.

[Hamscher *et al.*, 1992] W.C. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis.* Morgan Kaufmann, San Francisco, CA, 1992.

[Harnad, 1990] S. Harnad. The symbol grounding problem. *Physics*, 42:335–346, 1990.

[Hayes-Roth *et al.*, 1983] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, editors. *Building Expert Systems.* Addison-Wesley, 1983.

[Hayes-Roth *et al.*, 1992] F. Hayes-Roth, L.D. Erman, A. Terry, and B. Hayes-Roth. Distributed intelligent control and management: Concepts, methods and tools for developing DICAM applications. In *Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE-92)*, 1992.

[Hayes, 1973] P.J. Hayes. The frame problem and related problems in artificial intelligence. In Allen et al. [1990], pages 588–595.

[Hayes, 1975] P.J. Hayes. A representation for robot plans. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 181–188, 1975.

[Hayes, 1996] P. Hayes. A catalog of temporal theories. Technical Report Technical Report UIUC-BI-AI-96-01, The Beckman Institute, University of Illinois, 1996.

[Hertzberg, 1996] J. Hertzberg. On building a planning tool box. In Ghallab and Milani [1996], pages 3–18.

[Hess, 1996] D. Hess. Theater battle management command, control, communications, computer and intelligence architecture (TBM C4I) air operations center (AOC). Technical Report under contract for project 6970, U.S. Government and The Mitre Corporation, 1996.

[Hildum *et al.*, 1998] D.W. Hildum, N.M. Sadeh, T.J. Laliberty, J. McA'Nulty, S.E. Smith, D. Kjenstad, and A. Tseng. Blackboard agents for supporting mixed-initiative management of integrated process-planning and production-scheduling solutions across the supply chain. In *AIPS '98 Workshop on Interactive and Collaborative Planning*, pages 37–42, Carnegie Mellon University, 1998.

[Hintikka, 1962] J. Hintikka. *Knowledge and Belief.* Cornell University Press, Ithaca, NY, 1962.

[Hollingsworth, 1994] D. Hollingsworth. Workflow management coalition: The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition, Avenue Marcel Thirty 204, 1200 Brussels, Belgium, 1994.

[Holzner, 1998] S. Holzner. *XML Complete.* McGraw–Hill, New York, London, 1998.

[Hsia *et al.*, 1994] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, and C. Chen. Formal approach to scenario analysis. *IEEE Software*, March:33–40, 1994.

[IBM Corp., 1992] IBM Corp. Business system development method, introducing bsdm. Technical Report Reference No. GE19–5387, International Business Machines Corporation, IBM UK Ltd., May 1992.

[ISO, 1995] ISO. Product data representation and exchange: Part 49: Integrated generic resources: Process structure and properties. Technical Report ISO Standard 10303-49, International Standards Organization, 1995.

[Jackson, 1975] M.A. Jackson. *Principles of Program Design.* Academic Press, New York, USA, 1975.

[Jacobson *et al.*, 1992] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering.* Addison-Wesley, 1992.

[Jantke *et al.*, 1996] K.P. Jantke, O. Arnold, and T. Lehmann. Flexible simulation scenarios for real-time planning in dynamic environments. In J.H. Stewman, editor, *Florida AI Research Symposium (FLAIRS-96), Special Track on Real-Time Planning and Reacting*, pages 90–95, Key West, FL, USA, 1996. Florida AI Research Society.

[Jarvis and Winstanley, 1996a] P. Jarvis and G. Winstanley. Dynamically assessed and reasoned task (DART) networks. In *Proceedings of the Sixteenth Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems (ES-96), Cambridge, UK*, 1996.

[Jarvis and Winstanley, 1996b] P. Jarvis and G. Winstanley. Objects and objectives: the merging of object and planning technologies. In *Proceedings of the Fifteenth Workshop of the UK Planning and Scheduling Special Interest Group, Liverpool, UK*, 1996.

[Jarvis and Winstanley, 1998] P. Jarvis and G. Winstanley. Reducing the semantic gap between application domains and AI planning technology: a compilation based approach. In *Workshop on Knowledge Acquisition and Knowledge Elicitation, Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh, USA*, 1998.

[Jarvis *et al.*, 1999] P. Jarvis, J. Moore, J. Stader, A. Macintosh, A. Casson du Mont, and P. Chung. Exploiting AI technologies to realise adaptive workflow systems. In *Proceedings of the 1999 AAAI Workshop on Agent Based Systems in the Business Context*, 1999.

[Jarvis, 1997] P. Jarvis. *Integration of Classical and Model–Based Planning.* Ph.D. Thesis, School of Computing and Mathematical Sciences, University of Brighton, Sussex, UK, 1997.

[Jin *et al.*, 1998] Z. Jin, D. Bell, F.G. Wilkie, and D. Leahy. Automatically acquiring the requirements of business information systems by reusing business ontology. In *ECAI '98 Workshop on Applications of Ontologies and Problem-Solving Methods*, Brighton, England, 1998.

[Johnston and Adorf, 1992] A. Johnston and A. Adorf. Scheduling with neural networks: the case of the hubble space telescope. *Computers and Operations Research*, 19(3–4):209–240, 1992.

[Joslin and Pollack, 1996] D. Joslin and M.E. Pollack. Passive and active decision postponement in plan generation. In Ghallab and Milani [1996], pages 37–48.

[Joslin, 1996] D. Joslin. *Passive and active decision postponement in plan generation.* Ph.D. Thesis, University of Pittsburgh, Pittsburgh, PA, 1996.

[Kalfoglou and Robertson, 1999] Y. Kalfoglou and D. Robertson. Use of formal ontologies to support error checking in specifications. In *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modelling and Management (EKAW99)*, Dagsthul, Germany, 1999.

[Kambhampati and Hendler, 1992] S. Kambhampati and J. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

[Kambhampati *et al.*, 1995] S. Kambhampati, C. Knoblock, and Yang Q. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial–order planning. *Artificial Intelligence*, 76, 1995.

[Kambhampati, 1989] S. Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure-based approach.* Department of Computer Science, Ph.D. Thesis, University of Maryland, College Park, 1989.

[Kambhampati, 1994] S. Kambhampati. Multi-contributor causal structures for planning: a formalization and evaluation. *Artificial Intelligence*, 69(1-2):235–278, 1994.

[Kambhampati, 1996a] S. Kambhampati. Formalizing dependency directed backtracking and explanation-based learning in refinement search. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI-96, Portland, Oregon, Vol. 2*, Cambridge, Mass, 1996. MIT Press.

[Kambhampati, 1996b] S. Kambhampati. Refinement planning: Status and prospectus. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI-96, Portland, Oregon, Vol. 2*, pages 1331–1336, Cambridge, Mass, 1996. MIT Press.

[Kambhampati, 1997] S. Kambhampati. Refinement planning as a unifying framework for plan synthesis. *Artificial Intelligence Magazine*, 18(2), 1997.

[Karp *et al.*, 1995] P.D. Karp, K.L. Myers, and T. Gruber. The generic frame protocol. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, pages 768–774, 1995.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-92), Vienna, Austria*, 1992.

[Kazman *et al.*, 1996] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, 13(6):47–55, 1996.

[Kingston *et al.*, 1996] J. Kingston, N. Shadbolt, and A. Tate. CommonKADS models for knowledge based planning. Artificial Intelligence Application Institute AIAI-TR-199, University of Edinburgh, Edinburgh, Scotland, 1996.

[Kingston *et al.*, 1997] J.K. Kingston, A. Griffiths, and T.J. Lydiard. Multi-perspective modelling of the air campaign planning process. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Japan*, 1997.

[Kiritsis *et al.*, 1998] D. Kiritsis, P. Xirouchakis, and C. Gunther. Petri net representation for the process specification language - part 1: Manufacture process planning. CAD.CAM Laboratory Part 1, Swiss Federal Institute of Technology at Lausanne (EPFL), Switzerland, 1998.

[Knoblock(ed.), 1996] C. Knoblock(ed.). AI planning systems in the real world. *IEEE Expert Intelligent Systems and Their Application*, 11(6), December 1996.

[Knutilla *et al.*, 1998] A. Knutilla, C. Schlenoff, S. Ray, S. Polyak, A. Tate, S. Cheah, and R.C. Anderson. Process specification language: An analysis of existing representations. Technical Report NISTIR 6133, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

[Kondili *et al.*, 1993] E. Kondili, C.C. Pantelides, and R.W.H. Sargent. A general algorithm for short term scheduling of batch operations - I. MILP Formulation. *Comp. Chem. Engineering*, 17(2):211–227, 1993.

[Konolige and Nilsson, 1980] Kurt Konolige and Nils J. Nilsson. Multiple-agent planning systems. In *Proc. 1st AAAI*, pages 138–142, Stanford, CA, August 1980. Stanford University, William Kaufman.

[Konolige, 1982] Kurt Konolige. A first-order formalization of knowledge and action for a multi-agent planning system. In J. E. Hayes, D. Michie, and Y. Pao, editors, *Machine Intelligence 10*, pages 41–72. Ellis Horwood, Chichester, UK, 1982.

[Kotonya and Somerville, 1996] G. Kotonya and I. Somerville. Requirements engineering with viewpoints. *Soft. Engin. Journ.*, 11(1), 1996.

[Kowalski and Sergot, 1986] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[Kushmerick *et al.*, 1995] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[Kwok, 1995] C. Kwok. PDB reference manual. Technical Report Version 4.0, University of Washington, WA, USA, 1995.

[Langley and Drummond, 1990] P. Langley and M. Drummond. Toward an experimental science of planning. In K.P. Sycara, editor, *Proceedings workshop on innovative approaches to planning and scheduling approaches*, pages 109–114, San Diego, CA, 1990. Morgan Kaufmann Publishers, ISBN 1-55860-164-3.

[Lansky, 1987] A.L. Lansky. A representation of parallel activity based on events, structure, and causality. In M. Georgeff and A. Lansky, editors, *Reasoning About Plans*, pages 123–159. Morgan Kaufmann, Palo Alto, CA, 1987.

[Lansky, 1994] A. Lansky. Action-based planning. In *Proceedings of the Second International Conference on AI Planning Systems, Chicago, IL*, pages 110–115, 1994.

[Lassila, 1998] O. Lassila. Web metadata: A matter of semantics. *IEEE Internet Computing*, July–August, 1998.

[Lee and Billington, 1992] H.L. Lee and C. Billington. Supply chain management: Pitfalls and opportunities. *Sloan Management Review*, 33:65–73, 1992.

[Lee and Billington, 1993] H.L. Lee and C. Billington. Material management in decentralized supply chains. *Operations Research*, 41(5):835–847, 1993.

[Lee and Lai, 1991] J. Lee and K. Lai. What's in design rationale? *Human Comp. Inter.*, 6:251–280, 1991.

[Lee *et al.*, 1996] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, and G. Yost. The PIF process interchange format and framework version 1.1. Technical Report Working Paper No 194, MIT Center for Coordination Science, 1996.

[Lee *et al.*, 1998a] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, and G. Yost. The process interchange format and framework. *Knowledge Engineering Review*, 13(1), 1998.

[Lee *et al.*, 1998b] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, and G. Yost. PIF the process interchange format. In P. Bernusand K. Mertins and G. Schmidt, editors, *Handbook on Architectures of Information Systems*. Springer-Verlag, Berlin Heidelberg New York, 1998.

[Lee, 1990] J. Lee. SIBYL: A qualitative decision management system. In P.H. Winston and S. Shellard, editors, *Artificial Intelligence at MIT: Expanding Frontiers*, pages 104–133. MIT Press, Cambridge, MA, 1990.

[Lehrer, 1993] N. Lehrer. ARPI KRSL reference manual 2.0.2. Technical Report February, ISX Corporation, 1993.

[Lenat, 1995] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, November 1995.

[Levesque *et al.*, 1997] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R.B. Scherl. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31:59–83, 1997.

[Liebowitz and De Salvo, 1989] J. Liebowitz and D.A. De Salvo. *Structuring Expert Systems Domain, Design, and Development.* Prentice-Hall International, Englewood Cliffs, N.J., 1989.

[Lin and Shoham, 1991] F. Lin and Y. Shoham. Provably correct theories of action. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 349–354, 1991.

[London, 1978] P.E. London. Dependency networks as a representation for modelling in general problem solvers. Department of Computer Science TR-698, University of Maryland, 1978.

[Loveland, 1978] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis.* Fundamental Studies in Computer Science Vol. 6. North-Holland, Amsterdam, The Netherlands, 1978.

[Ludlow and Alguire, 1994] C.O. Ludlow and K.M. Alguire. Looking at O-Plan2 and SIPE-2 through missionaries and cannibals. In Burstein [1994], pages 453–464.

[Luger and Stubblefield, 1998] G. Luger and W. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* Addison Wesley Longman, Inc., 1998.

[Luke *et al.*, 1997] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In *Proceedings of The First International Conference on Autonomous Agents*, 1997.

[MacLean *et al.*, 1991] A. MacLean, R. Young, V. Bellotti, and T. Moran. Design space analysis: Bridging from theory to practice via design rationale. In *Proceedings of Esprit '91*, pages 720–730, Brussels, November 1991.

[Madni and Mi, 1997] A. Madni and P. Mi. IDEON specification in CORBA IDL. Technical Report ISTI–TM–7/97, Intelligent Systems Technology, Inc. (ISTI), Santa Monica, CA, 1997.

[Malone and Lee, 1990] T.W. Malone and J. Lee. Partially shared views: A scheme for communicating among groups that use different type hierarchies. *ACM Transactions on Information Systems*, 8(1), January 1990.

[Malone *et al.*, 1993] T.W. Malone, K. Crowston, J. Lee, and B.T. Pentland. Tools for inventing organizations: Towards a handbook of organizational processes. In *Proceedings of the 2nd IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises*, Morgantown, WV, 1993.

[Marcus (ed.), 1988] S. Marcus (ed.). *Automatic Knowledge Acquisition for Expert Systems.* Kluwer, 1988.

[Mayer *et al.*, 1992] R.J. Mayer, T.P. Cullinane, P.S. deWitte, W.B. Knappenberger, B. Perakath, and M.S. Wells. Information integration for concurrent engineering (IICE) IDEF3 process description capture method report. Technical Report AL-TR-1992-0057, Armstrong Laboratory, Logistics Research Division, Wright-Patterson AFB, OH 45433 USA, 1992.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the AAAI-91*, pages 634–639, Anaheim, CA, 1991.

[McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Allen et al. [1990], pages 393–435.

[McCarthy, 1980] John McCarthy. Circumscription—a form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McCluskey and Kitchin, 1998] T.L. McCluskey and D.E. Kitchin. OCLh: An object-centred language for HTN planning. School of computing and mathematics, University of Huddersfield, Huddersfield, UK, Submitted to ICTAI '98, 1998.

[McCluskey and Porteous, 1997] T.L. McCluskey and J.M. Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95(1):1–65, 1997.

[McDermott and Hendler, 1995] D. McDermott and J. Hendler. Planning: What it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence*, 76:1–16, 1995.

[McDermott, 1982] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[McDermott, 1985] D. McDermott. Reasoning about plans. In J. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World*, pages 269–317. Ablex Publishing Co., Norwood, NJ, 1985.

[McKay and Moore, 1991] K.N. McKay and J.B. Moore. Intelligent manufacturing management program: State of the art scheduling survey. Technical Report R–91–IMM–01, CAM–I, 1250 E. Copeland Road, Suite 500, Arlington, TX 76011 USA, 1991.

[Mertins and Schmidt, 1998] P. Bernusand K. Mertins and G. Schmidt, editors. *Handbook on Architectures of Information Systems.* Springer-Verlag, Berlin Heidelberg New York, 1998.

[Millet, 1995] G. Millet. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[Missiaen *et al.*, 1995] L. Missiaen, M. Bruynooghe, and M. Denecker. CHICA, a planning system based on event calculus. *The Journal of Logic and Computation*, 5(5):579–602, 1995.

[Moran and Carroll, 1996] T.P. Moran and J.M. Carroll, editors. *Design Rationale: Concepts, Techniques, and Use.* Lawrence Erlbaum Associates, 1996.

[Motta *et al.*, 1991] E. Motta, T. Rajan, J. Domingue, and M. Eisenstadt. Methodological foundations of KEATS, the knowledge engineers' assistant. *Knowledge Acquisition*, 3(1):21–47, 1991.

[Mowbray and Zahavi, 1995] T.J. Mowbray and R. Zahavi. *The essential CORBA: Systems integration using distributed objects.* Wiley, New York Chichester, 1995.

[Mullery, 1979] G. Mullery. CORE: A method for controlled requirements specification. In *Proceedings of the 4th International Conference on Software Engineering*, Munich, 1979.

[Musen, 1989] Mark A. Musen. Automated support for building and extending expert models. *Machine Learning*, 4:349–377, 1989.

[Myers and Wilkins, 1997] K.L. Myers and D.E. Wilkins. The act-editor user's guide: A manual for version 2.2. SRI International Artificial Intelligence Center, Stanford University, Menlo Park, CA, September 1997.

[Nado *et al.*, 1996] R. Nado, M. Chams, J. Delisio, and W. Hamscher. Comet: An application of model-based reasoning to accounting systems. In *Proceedings of the Eighth Innovative Applications of Artificial Intelligence*, pages 1482–1490, Menlo Park, CA, 1996. AAAI Press.

[Nau *et al.*, 1995] D.S. Nau, S.K. Gupta, and W.C. Regli. AI planning versus manufacturing-operation planning: a case study. In *International Joint Conference on Artificial Intelligence*, 1995.

[Navarro, 1996] T. Navarro. CDIF – integrated meta–model, project management planning and scheduling subject area. Technical Report EIA–PN3239, CASE Data Interchange Format Division, Electronic Industries Association, 1996.

[Nebel *et al.*, 1997] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *Fourth European Conference on Planning ECP '97*, pages 338–350, 1997.

[Neches *et al.*, 1991] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W.R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, Fall, 1991.

[Newell and Simon, 1963] A. Newell and H.A. Simon. GPS, a program that simulates human thought. In Allen et al. [1990], pages 59–66.

[Newell and Simon, 1972] A. Newell and H.A. Simon. *Human Problem Solving.* Prentice Hall, 1972.

[Newell and Simon, 1976] A Newell and H.A. Simon. Computer science as empirical enquiry: Symbols and search. *Communications of the Association for Computing Machinery*, 19:113–26, March 1976.

[Nilsson, 1984] N.J. Nilsson. Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1984.

[Nonaka, 1994] I. Nonaka. A dynamic theory of organizational knowledge creation. *Organization Science*, 5(1):14–37, 1994.

[Nunes de Barros *et al.*, 1996] L. Nunes de Barros, A. Valente, and R. Benjamins. Modeling planning tasks. In Drabble [1996], pages 11–18.

[Nunes de Barros *et al.*, 1998] L. Nunes de Barros, R. Benjamins, Y. Shahar, A. Tate, and A. Valente, editors. *Proceedings of the AIPS-98 workshop on Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice AAAI Technical Report WS-98-03*, Carnegie-Mellon University, June 1998.

[O'Hara *et al.*, 1992] K. O'Hara, N. Shadbolt, P. Laublet, M. Zacklad, and B. Leroux. Knowledge acquisition methodology. VITAL Project Report DD212, University of Nottingham, Nottingham, UK, 1992.

[Orgun and Ma, 1994] M.A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D. M. Gabbay and H. J. Ohlbach, editors, *Proc. of ICTL'94: The First International Conference on Temporal Logic*, pages 445–479, 1994.

[Orgun and Wadge, 1992] M.A. Orgun and W.W. Wadge. Theory and practice of temporal logic programming. In L.F. del Cerro and M. Penttonen, editors, *Intensional Logics for Programming.* Oxford University Press, 1992.

[Ousterhout, 1994] J.K. Ousterhout. *Tcl and the Tk Toolkit.* Addison Wesley Publishing Company, Inc., 1994.

[Pape, 1994] C. Le Pape. Implementation of resource constraints in ILOG schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.

[Parsons *et al.*, 1998] S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[Pease and Carrico, 1999] R.A. Pease and T.M. Carrico. Object Model Working Group (OMWG) Core Plan Representation. Technical Report Request for Comment, Version 4, Defense Advanced Research Projects Agency, 1999.

[Pednault, 1987] E. Pednault. Formulating mulitagent, dynamic-world problems in the classical planning framework. In Allen et al. [1990], pages 675–710.

[Penberthy and Weld, 1992] J.S. Penberthy and D.S. Weld. Ucpop: a sound, complete, partial order planer for adl. In *Proceedings Third International Conference on Principles of Knowledge Representation and Reasoning, Boston, MA*, 1992.

[Penberthy and Weld, 1994] J.S. Penberthy and D.S. Weld. Temporal planning with continuous change. In *Proceedings of AAAI-94*, pages 1010–1015, 1994.

[Pentland, 1994] B.T. Pentland. Process grammars: A generative approach to process redesign. Technical Report Working Paper No 178 3722-94, MIT Center for Coordination Science, 1994.

[Peot and Smith, 1996] M.A. Peot and D.E. Smith. Conditional nonlinear planning. In Tate [1996a], pages 210–217.

[Perez, 1996] A. Perez. Representing and learning quality-improving search control knowledge. In L. Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, Los Altos, CA, 1996. Morgan Kaufmann.

[Perini and Ricci, 1996] A. Perini and F. Ricci. An interactive planning architecture. In Ghallab and Milani [1996], pages 273–283.

[Petrie *et al.*, 1999] C. Petrie, S. Goldmann, and A. Raquet. Agent-based project management. In *Lecture Notes in AI 1500*. Springer–Verlag, 1999.

[Pinto and Reiter, 1993a] J. Pinto and R. Reiter. Adding a time line to the situation calculus. In *Proceedings of the Second AAAI Symposium on Logical Formilizations of Commonsense Reasoning*, pages 172–177, 1993.

[Pinto and Reiter, 1993b] J. Pinto and R. Reiter. Temporal reasoning in logic programming: a case for the situation calculus. In *Proceedings of the International Conference on Logic Programming*, pages 203–221. MIT Press, 1993.

[Pollack, 1992] M.E. Pollack. The uses of plans. *Artificial Intelligence*, 57(1):43–68, 1992.

[Pollack, 1996] M. Pollack. Planning in dynamic environments: The DIPART system. In Tate [1996a], pages 218–225.

[Pólya, 1945] G. Pólya. *How to Solve It.* Princeton University Press, Princeton, N.J., 1945.

[Polyak and Aitken, 1998] S. Polyak and S. Aitken. Manufacturing process interoperability scenario. Artificial Intelligence Applications Institute AIAI-PR-68, University of Edinburgh, Edinburgh, Scotland, 1998.

[Polyak and Tate, 1997] S. Polyak and A. Tate. Analysis of candidate PSL process/plan representations. Artificial Intelligence Applications Institute AIAI-PR-66, University of Edinburgh, Edinburgh, Scotland, 1997.

[Polyak and Tate, 1998] S. Polyak and A. Tate. Rationale in planning: Causality, dependencies, and decisions. *Knowledge Engineering Review*, 13(3):247–262, September 1998.

[Polyak and Tate, 1999] S. Polyak and A. Tate. A common process ontology for process-centred organisations. *Knowledge-Based Systems*, Submitted, 1999. Also available as: Institute of Representation and Reasoning (IRR), RP 930, University of Edinburgh, Scotland, 1998.

[Polyak *et al.*, 1998] S. Polyak, J. Lee, M. Gruninger, and C. Menzel. Applying the process interchange format (PIF) to a supply chain process interoperability scenario. In *ECAI '98 Workshop on Applications of Ontologies and Problem-Solving Methods*, pages 88–97, 1998.

[Polyak, 1997a] S. Polyak. Process plan representation scenario. Artificial Intelligence Applications Institute AIAI-PR-67, University of Edinburgh, Edinburgh, Scotland, 1997.

[Polyak, 1997b] S. Polyak. Requirements for a rich, shared plan representation. Department of Artificial Intelligence DP 187, University of Edinburgh, Edinburgh, Scotland, 1997.

[Polyak, 1998a] S. Polyak. Applying design space analysis to planning. In *Proceedings of the AIPS-98 workshop on Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice AAAI Technical Report WS-98-03*, Carnegie-Mellon University, June 1998.

[Polyak, 1998b] S. Polyak. The common process language. Department of Artificial Intelligence RP 933, University of Edinburgh, Edinburgh, Scotland, 1998.

[Polyak, 1998c] S. Polyak. Mapping timepoint-based constraints into interval relationships. Department of Artificial Intelligence RP 932, University of Edinburgh, Edinburgh, Scotland, 1998.

[Polyak, 1998d] S. Polyak. Process plan representation scenario. Artificial Intelligence Applications Institute (AIAI) AIAI-PR-67, University of Edinburgh, Edinburgh, Scotland, 1998.

[Polyak, 1998e] S. Polyak. A supply chain process interoperability demonstration using the process interchange format (PIF). Department of Artificial Intelligence RP 917, University of Edinburgh, Edinburgh, Scotland, 1998.

[Polyak, 1998f] S. T. Polyak. A supply chain process interoperability demonstration using the process interchange format (PIF). Department of Artificial Intelligence Report Number 918, University of Edinburgh, Edinburgh, Scotland, 1998.

[Polyak, 1999] S. Polyak. A common process methodology for engineering process domains. In D. Bustard, P. Kawalek, and M. Norris, editors, *Systems Modelling for Business Process Improvement (SMBPI)*. Artech House, 1999.

[Prerau, 1990] D.S. Prerau. *Developing and Managing Expert Systems.* Addison-Wesley, Reading, MA, 1990.

[Quinn *et al.*, 1996] J.B. Quinn, P. Anderson, and S. Finkelstein. Managing professional intellect: Making the most of the best. *Harvard Business Review*, March-April:71–80, 1996.

[Quintero, 1996] J. Quintero. REACTIVE PASCAL and the event calculus: a platform to program, reactive, rational agents. In U.C. Sigmund and M. Thielsche, editors, *Proceedings of the Workshop at FAPR'96: Reasoning about Actions and Planning in Complex Environments*, 1996.

[Rashid and Helal, 1997] Mosfeq Rashid and Sumi Helal. The CEDMOS reference architecture. Report CMI-136-97, MCC, Austin, TX, October 1997.

[Reece and Tate, 1994] G.A. Reece and A. Tate. Synthesizing protection monitors from causal structure. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 146–151, Chicago, IL, June 1994.

[Reece, 1994] G.A. Reece. *Characterization and Design of Competent Rational Execution Agents for Use in Dynamic Environments.* Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh, 1994.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Rich and Knight, 1991] E. Rich and M. Knight. *Artificial Intelligence.* McGraw-Hill Inc., 1991.

[Sacerdoti, 1974] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Sacerdoti, 1975] E.D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 206–214, 1975.

[Sadri and Kowalski, 1995] F. Sadri and R. Kowalski. Variants of the event calculus. In L. Sterling, editor, *Proceedings of the International Conference on Logic Programming.* MIT Press, 1995.

[Sander, 1987] K.F. Sander. *Microwave Components and Systems.* Addison-Wesley, Reading, Massachusetts, 1987.

[Sander, 1994] G. Sander. Graph layout throught the VCG tool. Technical Report Technical Report A03/94, Universit at des Saarlandes, FB 14 Informatik, 1994.

[Sander, 1995] G. Sander. Graph layout throught the VCG tool. In R. Tamassia and I.G. Tollis, editors, *DIMACS International Workshop GD'94, Lecture Notes in Computer Science*, pages 194–205. Springer Verlag, 1995.

[Schlenoff *et al.*, 1996] C. Schlenoff, A. Knutilla, and S. Ray. Unified process specification language: Requirements for modeling process. Technical Report NISTIR 5910, National Institute of Standards and Technology, Gaithersburg, MD, 1996.

[Schlenoff *et al.*, 1998] C. Schlenoff, A. Knutilla, S. Angster, S. Polyak, M. Gruninger, K. Jha, M. Ciocoiu, C. Menzel, and D. Kiritsis. Process specification language: An analysis of existing representations. Technical Report NISTIR December, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

[Schwalb and Vila, 1996] E. Schwalb and L. Vila. Logic programming with temporal constraints. In *Proceedings of the International Workshop on Temporal Representation and Reasoning*, pages 51–56, 1996.

[Schwalb and Vila, 1998] E. Schwalb and L. Vila. Temporal constraints: A survey. *Constraints: An International Journal*, 3(2/3), 1998.

[Shanahan, 1997a] M. Shanahan. *Solving the Frame Problem: A mathematical Investigation of the Common Sense Law of Inertia.* MIT Press, Cambridge, Mass., 1997.

[Shanahan, 1997b] M.P. Shanahan. Event calculus planning revisited. In *Proceedings of the 4th European Conference on Planning*, pages 390–402, 1997.

[Simmons *et al.*, 1998] R. Simmons, M. Veloso, and S. Smith, editors. *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Carnegie Mellon University, Pittsburgh, PA*, Menlo Park, CA, 1998. AAAI Press.

[Smith and Becker, 1997] S.F. Smith and M. Becker. An ontology for constructing scheduling systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, Stanford, CA, 1997. AAAI Press.

[Smith *et al.*, 1996] S.J.J. Smith, K. Hebbar, D.S. Nau, and I. Minis. Integrating electrical and mechanical design and process planning. In *IFIP Knowledge Intensive CAD Workshop*, Carnegie-Mellon University (CMU), September 16–18 1996.

[Smith, 1994] S.F. Smith. OPIS: A methodology and architecture for reactive scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling.* Morgan Kaufmann Publishers, 1994.

[Smith, 1997] S.J.J. Smith. *Task-Network Planning Using Total-Order Forward Search, and Applications to Bridge and to Microwave Module Manufacture.* Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park, 1997.

[Snowdon and Warboys, 1994] R.A. Snowdon and B.C. Warboys. An introduction to process-centered environments. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 1–8. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.

[Sommerville and Sawyer, 1997] I. Sommerville and P. Sawyer. *Requirements Engineering, A Good Practice Guide.* John Wiley and Sons Ltd., Chichester, West Sussex, England, 1997.

[Soutter, 1997] J. Soutter. *An Integrated Architecture for Operating Procedure Synthesis.* Ph.D. Thesis, Loughborough University, Loughborough, Leicestershire, UK, 1997.

[Srivastava *et al.*, 1997] B. Srivastava, A. Mali, and S. Kambhampati. A structured approach for synthesizing planners from specifications. In *12th IEEE Intl. Conf. on Automated Software Engineering*, Lake Tahoe, 1997. IEEE.

[Stallman and Sussman, 1977] R.M. Stallman and G.J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, October 1977.

[Steels, 1990] L. Steels. Components of expertise. *AI Magazine*, Summer 1990.

[Stefik *et al.*, 1983] M. Stefik, R. Balzer, J. Aikins, J. Benoit, L. Biernbaum, F. Hayes-Roth, and E. Sacerdoti. The architecture of expert systems. In F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, editors, *Building Expert Systems.* Addison-Wesley, 1983.

[Stefik, 1981] M. J. Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:141–169, 1981.

[Stephens and Whitehead, 1984] J. Stephens and R. Whitehead. The analyst – an expert system approach to requirements analysis. In *Proceeding of the third seminar on Application of Machine Intelligence to Defence Systems*, June 1984.

[Sussman, 1973] G. J. Sussman. HACKER: a computational model of skill acquisition. Memo 297, AI Lab, MIT, 1973.

[Sussman, 1974] G.J. Sussman. The virtuous nature of bugs. In *Proceedings of the AISB Summer Conference*, July 1974.

[Swartout and Gil, 1995] B. Swartout and Y. Gil. EXPECT: Explicit representations for flexible acquisition. In *Proceedings of the Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*, February 1995.

[Swartout and Gil, 1996] W.R. Swartout and Y. Gil. EXPECT: A user-centered environment for the development and adaptation of knowledge-based planning aids. In Tate [1996a], pages 250–258.

[Tate and Currie, 1984] A. Tate and K. Currie. The O-Plan task formalism workstation. Artificial Intelligence Applications Institute (AIAI) AIAI-TR-7, University of Edinburgh, 1984.

[Tate and Currie, 1985] A. Tate and K. Currie. The O-Plan task formalism workstation. In *Proceedings of the Third Workshop of the UK Alvey Programme's Planning Special Interest Group*, London, UK, 1985. Institute of Electrical Engineers.

[Tate and Drabble, 1995] A. Tate and B. Drabble. O-Plan's planworld viewers. In *Fourteenth UK Special Interest Group on Planning and Scheduling*, Wivenhoe House Conference Centre, Essex University, 1995.

[Tate *et al.*, 1990] A. Tate, J. Hendler, and M. Drummond. A review of AI planning techniques. In Allen et al. [1990], pages 26–49.

[Tate *et al.*, 1994a] A. Tate, B. Drabble, and J. Dalton. Task formalism manual. Artificial Intelligence Applications Institute AIAI-TF-Manual, University of Edinburgh, Edinburgh, UK ftp://ftp.aiai.ed.ac.uk/pub/documents/ANY/oplan-tf-manual.ps.gz, 1994.

[Tate *et al.*, 1994b] A. Tate, B. Drabble, and J. Dalton. The use of condition types to restrict search in an AI planner. In *Proceedings of the 12th National Conference on AI (AAAI-94)*, 1994.

[Tate *et al.*, 1994c] A. Tate, B. Drabble, and R. Kirby. O-Plan2: An architecture for command, planning and control. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.

[Tate *et al.*, 1996] A. Tate, B. Drabble, and J. Dalton. O-Plan: a knowledge-based planner and its application to logistics. In Tate [1996a], pages 259–266.

[Tate *et al.*, 1998a] A. Tate, J. Dalton, and J. Levine. Generation of multiple qualitatively different plan options. In Simmons et al. [1998], pages 171–179.

[Tate *et al.*, 1998b] A. Tate, S. Polyak, and P. Jarvis. TF Method: An initial framework for modelling and analysing planning domains,. AIPS '98 Workshop on Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice AAAI Technical Report WS-98-03, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1998.

[Tate *et al.*, 1999a] A. Tate, J. Dalton, J. Levine, and P. Jarvis. Using shared models of activity to underpin coalition planning. In *Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces*, Edinburgh, Scotland, May 1999.

[Tate *et al.*, 1999b] A. Tate, S. Polyak, and P. Jarvis. Knowledge acquisition for AI planning within the O-Plan project. In *Proceedings of the 1st Workshop of the PLANET Knowledge Acquisition Technical Coordination Unit*, Salford University, Salford, UK, April 1999.

[Tate, 1975] A. Tate. Interacting goals and their use. In *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI-75)*, Tbilisi, USSR, 1975.

[Tate, 1977] A. Tate. Generating project networks. In *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI-77)*, pages 888–893, Cambridge, MA, 1977.

[Tate, 1993a] A. Tate. Authority management – coordination between task assignment, planning and execution. In *Papers of the IJCAI-93 workshop on Knowledge-Based Production Planning, Scheduling and Control*, Chamberey, France, 1993.

[Tate, 1993b] A. Tate. Putting knowledge-rich plan representations to use. In *Papers of the 14th Machine Intelligence Workshop*, Tokyo, Japan, November 1993.

[Tate, 1994a] A. Tate. Mixed initiative planning in O-Plan2. In Burstein [1994].

[Tate, 1994b] A. Tate. Putting knowledge-rich process representations to use. *Journal of the IOPT Club for the Introduction of Process Technology*, 2(3):12–14, March 1994.

[Tate, 1995] A. Tate. Characterising plans as a set of constraints – the < I-N-OVA > model – a framework for comparitive analysis. *ACM Sigart Bulletin*, 6(1), January 1995.

[Tate, 1996a] A. Tate, editor. *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, Menlo Park, CA, 1996. AAAI Press.

[Tate, 1996b] A. Tate. KRSL-Plans, Appendix of Towards a Plan Ontology. *AI*IA Notizie (Quarterly Publication of the Associazione Italiana per l'Intelligenza Artificiale), Special Issue on Aspects of Planning Research*, 9(1):19–26, 1996.

[Tate, 1996c] A. Tate. Representing plans as a set of constraints, the <I-N-OVA> model. In Drabble [1996].

[Tate, 1996d] A. Tate. Towards a plan ontology. *AI*IA Notiziqe (Publication of the Associazione Italiana per l'Intelligenza Artificiale), Special Issue on Aspects of Planning Research*, 9(1):19–26, 1996.

[Tate, 1997] A. Tate. Mixed initiative interaction in O-Plan. In *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, Stanford, California, March 1997.

[Tate, 1998] A. Tate. Roots of SPAR - Shared Planning and Activity Representation. *The Knowledge Engineering Review*, 13(1):121–128, 1998.

[Tennison, 1999] J. Tennison. *Collaborative Knowledge Environments on the Internet.* Ph.D. Thesis, Department of Psychology, University of Nottingham, 1999.

[Thiébaux, 1995] S. Thiébaux. *Contribution à la planification sous incertitude et en temps contraint.* Ph.D. Thesis, Université de Rennes I, 1995.

[Tsang, 1993] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, San Diego, California, 1993.

[Upal and Elio, 1998] M.A. Upal and R. Elio. Learning to improve quality of the plans produced by partial order planners. AIPS '98 Workshop on Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice AAAI Technical Report WS-98-03, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1998.

[U.S. Army, 1997] U.S. Army. Staff organization and operations (field manual number 101-5). Technical Report 31-May-1997, Headquarters, Department of Army, Washington, DC, USA, 1997.

[Uschold and Gruninger, 1996] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), 1996.

[Uschold *et al.*, 1998] M.F. Uschold, S. Moralee, M. King, and Y. Ziorgios. The enterprise ontology. *Knowledge Engineering Review*, 13(1), 1998.

[Ushold *et al.*, 1998] M. Ushold, M. Healy, K. Williamson, P. Clark, and S. Woods. Ontology reuse and application. In N. Guarino, editor, *Formal Ontology in Information Systems*, pages 179–192. IOS Press, Amsterdam, Netherlands, 1998.

[Valente *et al.*, 1996] A. Valente, W.R. Swartout, and Y. Gil. A representation and library for objectives in air campaign plans. Technical report, University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, 1996.

[Valente, 1995] A. Valente. Knowledge-level analysis of planning systems. SIGART *Bulletin*, 6(1), 1995.

[van der Spek and de Hoog, 1996] R. van der Spek and R. de Hoog. A framework for knowledge management methodology. In K. Wiig, editor, *Volume 3; Knowledge Management methods, Practical approaches to managing knowledge*, pages 379–393, 5211 Vicksburg Dr, Arlington, TX, 1996. Schema Press, ISBN 0-9638925-2-5.

[van Lamsweerde and Letier, 1998] A. van Lamsweerde and E. Letier. Integrating obstacles in goal-driven requirements engineering. In *Proceedings (ICSE'98) – 20th International Conference on Software Engineering, (IEEE-ACM)*, Kyoto, 1998.

[Veloso *et al.*, 1995] M. Veloso, J. Carbonell, M.A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[Veloso *et al.*, 1998] M.M. Veloso, M.E. Pollack, and M.T. Cox. Rationale-based monitoring for planning in dynamic environments. In Simmons et al. [1998], pages 171–179.

[Veloso, 1996] M.M. Veloso. Toward mixed-initiative rationale-supported planning. In Tate [1996a], pages 277–282.

[Vere, 1991] S. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5, 1991.

[Walther, 1985] C. Walther. *A Many-Sorted Calculus Based on Resolution and Paramodulation*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1985.

[Wang, 1996] X. Wang. Planning while learning operators. In Drabble [1996], pages 229–236.

[Weld, 1994] D.S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[Weld, 1996] Daniel S. Weld. Planning-based control of software agents. In Brian Drabble, editor, *Proc. 3rd International Conference on Artificial Intelligence Planning Sytems*, pages 268–274, Edinburgh, Scotland, May 1996. AAAI Press.

[Weld, 1999] D.S. Weld. Recent advances in AI planning. *AI Magazine*, To Appear, 1999.

[WfMC, 1994] WfMC. Workflow Management Coalition Glossary, Workflow Management Coalition specification. Technical Report November '94, Workflow Management Coalition, http://www.wfmc.org, 1994.

[Wickler, 1999] G. Wickler. *Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation.* Ph.D. Thesis, School of Artificial Intelligence, University of Edinburgh, 1999.

[Wielinga *et al.*, 1992] B. Wielinga, W. van de Velde, G. Schriber, and H. Akkermans. The KADS knowledge modelling approach. In *Proceedings of the Japanese Knowledge Acquisition Workshop*, 1992.

[Wilkins and Desimone, 1994] D.E. Wilkins and R.V. Desimone. Applying an AI planner to military operation planning. In M. Zweben and M. Fox, editors, *Intelligent Scheduling.* Morgan Kaufmann Publishing Co., 1994.

[Wilkins and Myers, 1995] D.E. Wilkins and K.L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6):269–301, 1995.

[Wilkins and Myers, 1998] D.E. Wilkins and K.L. Myers. A multiagent planning architecture. In Simmons et al. [1998], pages 154–162.

[Wilkins *et al.*, 1995] D. E. Wilkins, K.L. Myers, J.D. Lowrance, and L.P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227, 1995.

[Wilkins, 1984] D. E. Wilkins. Domain-independent planning: representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[Wilkins, 1988] D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988.

[Wilson, 1984] A.C.M. Wilson. *Information for Planning.* M.Sc. Thesis, Department of Artificial Intelligence, University of Edinburgh, UK, 1984.

[Wooldridge and Jennings, 1998] M. Wooldridge and N. R. Jennings. Pitfalls of agent-oriented development. In *Proc. of the 2nd International Conference on Autonomous Agents (Agents-98)*, pages 385–391, Minneapolis, MN, USA, 1998.

[Wooldridge *et al.*, 1999] M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proc. 3rd Int Conference on Autonomous Agents (Agents-99)*, Seattle, WA, 1999.

[Zweben and Fox, 1994] M. Zweben and M. Fox, editors. *Intelligent Scheduling.* Morgan Kaufmann Publishing Co., 1994.

Appendix A

Common Process Ontology Reference

A.1 Ontolingua Source for CPO

This section presents version 1.0 of the Common Process Ontology (CPO) using ontolingua [Gruber, 1993]. The most up-to-date ontolingua code for this CPO core as well as the extensions described in this paper is available at the CPF homepage: http://www.aiai.ed.ac.uk/~oplan/cpf.

```
;;; -- Steve Polyak 9 April 1998 -- created outline.
;;; -- Steve Polyak 21 April 1998 -- major changes to the structure.
;;; -- Steve Polyak 30 November 1998 -- rewrite for CPF.
;;; -- Steve Polyak 9 June 1999 -- final changes for initial version.

(in-package "ONTOLINGUA-USER")

(define-theory CPO (frame-ontology slot-constraint-sugar cpo-expressions)
  "This is version 1.0 of the Common Process Ontology (CPO).  CPO is
  used to specify process domain knowledge or to specify an individual
  process or plan. 3-CPO: The three elements of the Common Process
  Ontology: Meta-Ontology, Constraint-Ontology, Object-Ontology.
  Meta-Ontology provides fundamental ontological elements used to
  describe the others and the assumptions behind the description. The
  subject of the Object-Ontology is activity processes along with core
  activity-relatable objects.  The Constraint-Ontology provides the
  restrictions on the space of possible behavior."
  :issues ((:copyright "Copyright (c) 1998 Steve Polyak")))

(in-theory 'CPO)

;;; ***********************************************************************
;;; CPO META-ONTOLOGY
;;; ***********************************************************************
```

```
;;; Ontologies will commonly have a top-level class such as the
;;; following. They are more for organization rather than meaning, but
;;; are useful for assimilating a small ontology into a more
;;; comprehensive ontology.

;;;CPO-Entity

(define-class CPO-ENTITY (?entity)
  "A CPO-ENTITY is a fundamental thing in the domain being modelled.
   A CPO-ENTITY may participate in relationships with other entities."
  :def (individual-thing ?entity)
  :axiom-def (subclass-partition
              CPO-ENTITY
              (setof cpo-process
     cpo-activity-relatable-object
     cpo-domain-level
     cpo-node
     cpo-timepoint
     cpo-constraint)))

(define-relation ENTITY.ISA (?class1 ?class2)
  "This relation between classes is provided to allow subclassing to
  be defined within a process specification. The subclass-of relation is
  defined in the frame ontology as: a class1 is a subclass of parent
  class2 if and only if every instance of class1 is also an instance of
  class2."
  :def (and (class ?class1)
    (class ?class2)
    (subclass-of ?class1 ?class2)))

;;;CPO-String

(define-class CPO-STRING (?string)
  "The most general class of CPO text objects."
  :def (string ?string))

;;; CPO-Set

(define-frame CPO-Set
  "The most general Set-Class in the Ontology. Based on the Formal
  Enterprise Ontology (v.1.1) EO-SET. Every instance of Set-Class is a
  subclass of CPO-Set. This is an abstract class provided mainly for
  convenience, so it is easy to see what all the Set-Classes are. It is
  up to Ontology developers, users and maintainers to make sure each
  instance of Set-Class is declared to be a subclass of CPO-Set."

  :iff-def (and (Instance-Of Set-Class CPO-Set)
```

```
(SubClass-Of CPO-Entity CPO-Set))
  :axioms
  (<=> (CPO-Set ?x)
       (Exists (?sc)
       (and (Instance-Of ?sc Set-Class)
    (Instance-Of ?x ?sc)))))

(define-frame Set-Class
  "Set-Class is a meta-Class.  Its instances are special kinds of classes,
  all of whose instances are themselves sets (not Classes) such that
  every member of such a set is specified to be a member of a certain Class."
  :own-slots
   ((Subclass-Of Class))
  :axioms
  (<=>
   (Set-Class ?set-of-things)
   (Exists (?thing)
   (and (Class ?thing)
(forall (?things)
(<=> (instance-of ?things ?set-of-things)
     (and (set ?things)
  (forall (?x)
  (=> (member ?x ?things)
      (instance-of ?x ?thing))))))))))
;;; CPO-Boolean

(define-class CPO-BOOLEAN (?flag)
  :iff-def (member ?flag
                  (setof true false)))

;;; *********************************************************************
;;; CPO OBJECT-ONTOLOGY
;;; *********************************************************************

;;; Cpo-Process

(define-class CPO-PROCESS (?x)
  "A process is a specification of behaviour."
  :def (and (cpo-entity ?x)
            (action ?x)  ;;; defined in basic sit. calc.
    (has-one ?x process.pattern)
            (has-one ?x process.start-timepoint)
            (has-one ?x process.finish-timepoint))
  :issues   (("extensions required for levels and phases.")))

(define-function PROCESS.PATTERN (?process) :-> ?pattern
  "The unifiable pattern which can be used to match node patterns."
```

```
  :def (and (cpo-process ?process)
            (cpo-string ?pattern)))

(define-function PROCESS.START-TIMEPOINT (?proc) :-> ?tp
  :def (and (cpo-process ?proc)
            (cpo-timepoint ?tp))
  :axioms
  (forall (?s)
          (=  ?tp (start (do ?proc ?s))))
  (forall ?proc ?t1)
  (<=> (= (process.start-timepoint ?proc) ?t1)
       (exists (?t2)
               (OccursT ?proc ?t1 ?t2))))

(define-function PROCESS.FINISH-TIMEPOINT (?proc) :-> ?tp
  :def (and (cpo-process ?proc)
            (cpo-timepoint ?tp))
  :axioms
  (forall (?s)
          (=  ?tp (end (do ?proc ?s) ?proc)))

  (forall ?proc ?t1)
  (<=> (= (process.end-timepoint ?proc) ?t1)
       (exists (?t2)
               (OccursT ?proc ?t2 ?t1))))

(define-relation PROCESS.ACTIVITY-SPEC (?process ?spec)
  "The specification of the activity. Note that there can be more than
   one specification attached to a process."
  :def (and (cpo-process ?process)
            (cpo-activity-specification ?spec)))

(define-function PROCESS.EXPANDS (?process) :-> ?node
  "The particular node that a process expands."
  :def (and (cpo-process ?process)
            (cpo-node ?node)))

;;; Cpo-Plan

(define-class CPO-PLAN (?x)
  "A process which is designed for some objectives is termed a plan."
  :def (and (cpo-process ?x)
    (has-one ?x plan.objective-spec)))

(define-relation PLAN.OBJECTIVE-SPEC (?plan ?spec)
  "The specification of the plan objectives."
  :def (and (cpo-plan ?plan)
```

```
            (cpo-objective-specification ?spec)))

;;; CPO-Node

(define-class CPO-NODE (?x)
  :def (and (cpo-entity ?x)))

;;; CPO-Activity

(define-class CPO-ACTIVITY (?x)
  :def (and (cpo-node ?x)
            (action ?x)
            (has-one ?x activity.pattern)
            (has-one ?x activity.begin-timepoint)
            (has-one ?x activity.end-timepoint))
  :issues   (("actions can be in phases or in levels")
     ("activities create, require, destroy, modify, etc.")))

(define-function ACTIVITY.PATTERN (?act) :-> ?pattern
  :def (and (cpo-activity ?act)
            (cpo-string ?pattern)))

(define-function ACTIVITY.BEGIN-TIMEPOINT (?act) :-> ?tp
  :def (and (cpo-activity ?act)
            (cpo-timepoint ?tp))
  :axioms
  (forall (?s)
          (=  ?tp (start (do ?act ?s))))
  (forall ?act ?t1)
  (<=> (= (activity.begin-timepoint ?act) ?t1)
       (exists (?t2)
               (OccursT ?act ?t1 ?t2))))

(define-function ACTIVITY.END-TIMEPOINT (?act) :-> ?tp
  :def (and (cpo-activity ?act)
            (cpo-timepoint ?tp))
  :axioms
  (forall (?s)
          (=  ?tp (end (do ?act ?s) ?act)))

  (forall ?a ?t1)
  (<=> (= (activity.end-timepoint ?a) ?t1)
       (exists (?t2)
               (OccursT ?a ?t2 ?t1))))

(define-function ACTIVITY.EXPANSION (?node) :-> ?proc
  "Indicates the process which expands this node."
```

```
  :def (and (cpo-activity ?node)
            (cpo-process ?proc))
  :axioms
  ;;; the expanded process is temporally bounded by the node
  (forall ?node ?proc
  (=> (= activity.expansion(?node) ?proc)
      (and (= begin.timepoint(?node) process.start-timepoint(?proc))
   (= end.timepoint(?node) process.finish-timepoint(?proc)))))
  ;;; expansion and expands imply each other
  (forall ?node ?proc
  (<=> (= activity.expansion(?node) ?proc)
       (= process.expands(?proc) ?node)))
  ;;; there can be only one defined expansion
  (forall ?node ?proc1
  (=> (= activity.expansion(?node) ?proc1)
      (not (exists (?proc2) (= activity.expansion(?node) ?proc2))))))

(define-relation CPO-SUBACTION (?act1 ?act2)
  :def (and (cpo-activity ?act1)
            (cpo-activity ?act2)
    (subaction@complex-actions ?act1 ?act2))
  :issues (("Need to resolve subaction with include node in an
             activity spec.")))

(define-class CPO-ACTION (?x)
  :def (cpo-activity ?x))

(define-relation CPO-PRIMITIVE (?act1)
  :def (and (cpo-action ?act1)
    (primitive@complex-actions ?act1))
  :issues (("This needs some work.")))

(define-class CPO-EVENT (?x)
  :def (and (cpo-activity ?x)
    (forall ?agent
    (not (agent.performs-act ?agent ?x)))))

(define-class CPO-OTHER-NODE (?x)
  :def (cpo-node ?x))

(define-class CPO-DUMMY-NODE (?x)
  :def (cpo-other-node ?x))

(define-class CPO-START (?x)
  :def (and (cpo-dummy-node ?x)
    (has-one ?x start.timepoint)))
```

```
(define-class CPO-FINISH (?x)
  :def (and (cpo-dummy-node ?x)
    (has-one ?x finish.timepoint)))

(define-class CPO-BEGIN (?x)
  :def (and (cpo-dummy-node ?x)
    (has-one ?x start.timepoint)))

(define-class CPO-END (?x)
  :def (and (cpo-dummy-node ?x)
    (has-one ?x finish.timepoint)))

(define-function START.TIMEPOINT (?node) :-> ?tp
  "Returns start timepoint for instantaneous nodes."
  :def (and (cpo-dummy-node ?node)
            (cpo-timepoint ?tp))

(define-function FINISH.TIMEPOINT (?node) :-> ?tp
  "Returns finish timepoint for instantaneous nodes."
  :def (and (cpo-dummy-node ?node)
            (cpo-timepoint ?tp))

;;; Cpo-Timepoint

(define-class CPO-TIMEPOINT (?x)
  "A CPO-TIMEPOINT is an entity that represents a specific, instananeous
   point along a time line which is an infinite sequence of time points."
  :def (and (time ?x) (cpo-entity ?x))
  :issues (("This needs to be properly connected to Pinto
             and Reiter's work.")
   ("Need to work out time interval note duration in complex act.")
   ("How about a CPO-Metric-Token?")))

;;; Cpo-Activity-Relatable-Object

(define-class CPO-ACTIVITY-RELATABLE-OBJECT (?x)
  "An activity-relatable-object is an abstract class used to
   group the objects which have a direct relationship to activities."
  :def (and (cpo-entity ?x)))

(define-function OBJECT.NAME (?object) :-> ?name
  :def (and (cpo-activity-relatable-object ?object)
            (cpo-string ?name)))

;;; Cpo-Agent

(define-class CPO-AGENT (?agent)
```

```
  "A cpo-agent is an entity that can perform behaviour, hold purpose(s),
   and have capabilities."
  :def (cpo-activity-relatable-object ?agent))

(define-instance ENVIRONMENT (cpo-agent)
  "There is a predefined AGENT called the environment. It can only
  establish enforced constraints and cannot intend, desire, or synthesize
  constraints."
  :axioms
  (forall ?objective ?plan
  (=> (and (cpo-objective ?objective) (cpo-plan ?plan))
      (not (agent.has-preference environment ?objective ?plan)))))

(define-relation AGENT.HAS-CAPABILITY (?agent ?capability)
  "This is a general mechanism for linking to capabilities."
  :def (and (cpo-agent ?agent)
            (cpo-string ?capability)))

(define-relation AGENT.HAS-REQUIREMENT (?agent ?objective ?plan)
  "This is the mechanism for an agent to enforce a constraint"
  :def (and (cpo-agent ?agent)
            (cpo-objective-constraint ?objective)
            (cpo-plan ?plan)
            (= soft-hard-information(?objective) HARD)))

(define-relation AGENT.HAS-PREFERENCE (?agent ?objective ?plan)
  "This is the mechanism for an agent to desire a constraint"
  :def (and (cpo-agent ?agent)
            (cpo-preference ?pref)
            (cpo-plan ?plan)
            (= soft-hard-information(?objective) SOFT)))

(define-relation AGENT.HAS-REQUIREMENT (?agent ?objective)
  "This is the mechanism for an agent to enforce a constraint"
  :def (and (cpo-agent ?agent)
            (cpo-objective-constraint ?objective)
            (= soft-hard-information(?objective) HARD)))

(define-relation AGENT.HAS-PREFERENCE (?agent ?objective)
  "This is the mechanism for an agent to desire a constraint"
  :def (and (cpo-agent ?agent)
            (cpo-objective-constraint ?objective)
            (= soft-hard-information(?objective) SOFT)))

(define-relation AGENT.HAS-PLAN (?agent ?plan)
  :def (and (cpo-agent ?agent)
            (cpo-plan ?plan))
```

```
  :issues ("need to relate having a plan to agent req/pref?"))

(define-relation AGENT.PERFORMS-ACT (?agent ?action)
  :def (and (cpo-agent ?agent)
            (cpo-action ?action)))

(define-relation AGENT.PERFORMS-PROC (?agent ?proc)
  :def (and (cpo-agent ?agent)
            (cpo-process ?proc)))

;;; CPO-Domain-Level

(define-class CPO-DOMAIN-LEVEL (?x)
  "A domain level is a partition of process specifications in a domain."
  :def (and (cpo-entity ?x)
    (has-one ?x domain-level.label)
    (has-one ?x domain-level.number)))

(define-function DOMAIN-LEVEL.LABEL (?level) :-> ?label
  "This is a user-readable description of the level."
  :def (and (cpo-domain-level ?level)
            (cpo-string ?label)))

(define-function DOMAIN-LEVEL.NUMBER (?level) :-> ?number
  "This is a property which may be used to order levels."
  :def (and (cpo-domain-level ?level)
            (integer ?number)))

(define-relation DOMAIN-LEVEL.CONTAINS (?level ?process)
  :def (and (cpo-domain-level ?level)
            (cpo-process ?process)))

;;; ******************************************************************
;;; CPO CONTRAINT-ONTOLOGY
;;; ******************************************************************

(define-class CPO-ACTIVITY-SPECIFICATION (?act-spec)
  "A Set-Class all of whose instances are sets whose members are all of
   Class CPO-CONSTRAINT."
  :iff-def
  (and (CPO-Set ?act-spec)
       (and (Set ?act-spec)
    (forall (?x)
    (=> (Member ?x ?act-spec)
(Instance-Of ?x CPO-CONSTRAINT)))))
  :issues (("This is a special Set-Class")))
```

```
(define-class CPO-OBJECTIVE-SPECIFICATION (?obj-spec)
  "A Set-Class all of whose instances are sets whose members are all of
   Class CPO-OBJECTIVE-CONSTRAINT."
  :iff-def
  (and (CPO-Set ?obj-spec)
       (and (Set ?obj-spec)
    (forall (?x)
    (=> (Member ?x ?obj-spec)
(Instance-Of ?x CPO-OBJECTIVE-CONSTRAINT)))))
  :issues (("This is a special Set-Class")))

;;; CPO-Constraint

(define-class CPO-CONSTRAINT (?x)
  "A constraint expresses an assertion that can be evaluated with
   respect to a given process as something that may hold and can be
   elaborated in some language. Note the added-by Rel provides knowledge
   of which agent synthesized the constraint."

  :def (and (cpo-entity ?x))
  :axiom-def (exhaustive-subclass-partition
              CONSTRAINT
              (setof cpo-issue-constraint
     cpo-node-constraint
                     cpo-other-constraint)))

(define-function CONSTRAINT.ADDED-BY (?x) :-> ?agent
  :def (and (cpo-constraint ?x)
            (cpo-agent ?agent)))

(define-function CONSTRAINT.EXPRESSION (?x) :-> ?exp
  :def (and (cpo-constraint ?x)
            (relation ?exp)))

;;; CPO-Constraint-Type

(define-class CPO-CONSTRAINT-TYPE (?type)
  "This is just an abstract class with represents two possible const.
   types: soft and hard."
  :iff-def (member ?type
                   (setof 'soft 'hard)))

(define-instance SOFT (cpo-constraint-type))
(define-instance HARD (cpo-constraint-type))

(define-function CONSTRAINT.SOFT-HARD-INFORMATION (?x) :-> ?type
  :def (and (cpo-constraint ?x)
```

```
            (cpo-constraint-type ?type)))

;;; CPO-Objective-Constraint

(define-class CPO-OBJECTIVE-CONSTRAINT (?x)
  "OBJECTIVE-CONSTRAINTS impose restrictions over a set of
   world states or require particular activities to be performed."
  :def (and (constraint ?x)
            (has-one ?x constraint.expression)
    (Instance-Of constraint.expression(?x) cpo-objective-expression))

;;; CPO-Issue-Constraint

(define-class CPO-ISSUE-CONSTRAINT (?x)
  "An issue is an oustanding aim, preference, task, flaw or
   other issue which remains to be addressed by the process. Issues provide
   implied constaints on the real world behaviour specified by the
   process. Issues are represented by a verb, zero, one or more noun phrases
   and zero, one or more qualifiers."
  :def (and (constraint ?x)
            (has-one ?x constraint.expression)
    (Instance-Of constraint.expression(?x) cpo-issue-expression))

;;; CPO-Node-Constraint

(define-class CPO-NODE-CONSTRAINT (?x)
  :def (cpo-constraint ?x)
  :axiom-def (exhaustive-subclass-partition
              CPO-NODE-CONSTRAINT
              (setof cpo-include-constraint
                     cpo-not-include-constraint)))

(define-class CPO-INCLUDE-CONSTRAINT (?x)
  :def (and (cpo-node-constraint ?x)
            (has-one ?x include-node))
  :axioms
  ;;; included activities are temporally contained within a process
  (forall ?act-spec ?proc ?node
  (=> (and (cpo-activity-specification ?act-spec)
   (cpo-process ?proc)
   (member ?x ?act-spec)
   (activity-spec ?proc ?act-spec)
   (= include-node(?x) ?node)
   (cpo-activity ?node))
      (and (< process.start-timepoint(?proc) begin.timepoint(?node))
   (< end.timepoint(?node) process.finish-timepoint(?proc)))))
```

```
  ;;; included start/begin nodes are fixed at the process start time
  (forall ?act-spec ?proc ?node
  (=> (and (cpo-activity-specification ?act-spec)
   (cpo-process ?proc)
   (member ?x ?act-spec)
   (activity-spec ?proc ?act-spec)
   (= include-node(?x) ?node)
   (or (cpo-start ?node) (cpo-begin ?node)))
      (= process.start-timepoint(?proc) start.timepoint(?node))))

  ;;; included finish/end nodes are fixed at the process finish time
  (forall ?act-spec ?proc ?node
  (=> (and (cpo-activity-specification ?act-spec)
   (cpo-process ?proc)
   (member ?x ?act-spec)
   (activity-spec ?proc ?act-spec)
   (= include-node(?x) ?node)
   (or (cpo-finish ?node) (cpo-end ?node)))
      (= process.finish-timepoint(?proc) finish.timepoint(?node)))))

(define-function INCLUDE-NODE (?x) :-> ?node
  :def (and (cpo-include-constraint ?x)
            (cpo-node ?node)))

(define-class CPO-NOT-INCLUDE-CONSTRAINT (?x)
  :def (and (cpo-node-constraint ?x)
    (has-one ?x not-include-node))
  :axioms
  ;;; not-include prevents mirror include node constraints
  (forall ?act-spec ?node
  (=> (and (cpo-activity-specification ?act-spec)
   (member ?x ?act-spec)
   (= not-include-node(?x) ?node))
      (not (exists ?y
   (and (cpo-include-constraint ?y)
(= include-node(?y) ?node)
(member ?y ?act-spec)))))))

(define-function NOT-INCLUDE-NODE (?x) :-> ?node
  :def (and (cpo-not-include-constraint ?x)
            (cpo-node ?node)))

;;; CPO-Other-Constraint

(define-class CPO-OTHER-CONSTRAINT (?x)
  :def (and (constraint ?x)
    (has-one ?x constraint.expression))
```

```
  :axiom-def (exhaustive-subclass-partition
              CPO-OTHER-CONSTRAINT
              (setof cpo-ordering-constraint
                     cpo-variable-constraint
                     cpo-auxiliary-constraint)))

;;; CPO-Ordering-Constraint

(define-class CPO-ORDERING-CONSTRAINT (?x)
  "It is possible to specify temporal relationships directly between
   timepoints, and through the association of timepoints with the begin
   and end of an activity, between activities themseleves. (e.g.
   before(tp1,tp2), equal(tp1,tp2))"
  :def (and (cpo-other-constraint ?x)
            (has-one ?x constraint.expression)
    (Instance-Of constraint.expression(?x) cpo-ordering-expression)))

;;; Cpo-Variable-Constraint

(define-class CPO-VARIABLE-CONSTRAINT (?x)
  "A relationship such as codesignation between entity variables
   non-co-designation, possibly others such as type membership
   general restriction facilities, ranges, etc."
  :def (cpo-other-constraint ?x))

;;; Cpo-Auxiliary-Constraint

(define-class CPO-AUXILIARY-CONSTRAINT (?x)
  :def (cpo-other-constraint ?x)

  :axiom-def (exhaustive-subclass-partition
              CPO-AUXILIARY-CONSTRAINT
              (setof cpo-authority-constraint
                     cpo-world-state-constraint
                     cpo-resource-constraint
                     cpo-spatial-constraint
                     cpo-misc-constraint)))

(define-class CPO-AUTHORITY-CONSTRAINT (?x)
  :def (cpo-auxiliary-constraint ?x))

(define-class CPO-WORLD-STATE-CONSTRAINT (?x)
  :def (cpo-auxiliary-constraint ?x))

;;; CPO-Input-Constraint

(define-class CPO-INPUT-CONSTRAINT (?x)
```

```
  "It is a temporal constraint which may or may not be satisfied
   immediately before the given timepoint."
  :def (cpo-world-state-constraint ?x)

;;; CPO-Output-Constraint

(define-class CPO-OUTPUT-CONSTRAINT (?x)
  "It is a temporal constraint which may or may not be satisfied
   immediately after the given timepoint."
  :def (cpo-world-state-constraint ?x)

;;; CPO-Range-Constraint

(define-class CPO-RANGE-CONSTRAINT (?x)
  "It is a temporal constraint involving two timepoints,
   the constraint should hold at all times between the two points."
  :def (cpo-world-state-constraint ?x)

;;; CPO-Metric-Constraint

(define-class CPO-METRIC-CONSTRAINT (?x)
  "It is a temporal constraint involving one timepoint,
   to relate a given timepoint to an actual time or calendar ref."
  :def (cpo-ordering-constraint ?x))

(define-class CPO-ALWAYS-CONSTRAINT (?x)
  :def (cpo-world-state-constraint ?x)
  :issues (("Quantify over all states and show that this is always true")))

(define-class CPO-RESOURCE-CONSTRAINT (?x)
  :def (cpo-auxiliary-constraint ?x)
  :axiom-def (subclass-partition
              CPO-RESOURCE-CONSTRAINT
              (setof cpo-agent-constraint)))

(define-class CPO-AGENT-CONSTRAINT (?x)
  :def (cpo-resource-constraint ?x))

(define-class CPO-SPATIAL-CONSTRAINT (?x)
  :def (cpo-auxiliary-constraint ?x))

(define-class CPO-MISC-CONSTRAINT (?x)
  :def (cpo-auxiliary-constraint ?x)

  :axiom-def (exhaustive-subclass-partition
              CPO-MISC-CONSTRAINT
              (setof cpo-quality-constraint
```

```
                     cpo-annotation-constraint)))

(define-class CPO-QUALITY-CONSTRAINT (?x)
  :def (cpo-misc-constraint ?x))

(define-class CPO-ANNOTATION-CONSTRAINT (?x)
  :def (cpo-misc-constraint ?x))

;;; End File
```

A.2 CPO Sort Table

The following table presents the abbreviations used in this thesis for referring to various core and extended CPO sort/class types (see Section 3.3.1).

$\mathcal{A}$	Activity	
$\mathcal{A}ro$	Activity-relatable object	
$\mathcal{A}s$	Activity specification	
$\mathcal{A}ct$	Action	
$\mathcal{C}$	Constraint	
	$\mathcal{C}_{alw}$	Always constraint
	$\mathcal{C}_{ann}$	Annotation constraint
	$\mathcal{C}_{aux}$	Auxiliary constraint
	$\mathcal{C}_{inc}$	Include constraint
	$\mathcal{C}_{inp}$	Input constraint
	$\mathcal{C}_{iss}$	Issue constraint
	$\mathcal{C}_{ord}$	Ordering constraint
	$\mathcal{C}_{out}$	Output constraint
	$\mathcal{C}_{res}$	Resource constraint
	$\mathcal{C}_{var}$	Variable constraint
$\mathcal{C}rt$	Criteria	
$\mathcal{D}$	Domain level	
$\mathcal{D}r$	Decision rationale	
$\mathcal{E}$	Entity	
$\mathcal{E}vt$	Event	
$\mathcal{E}xp$	Expression	
$\mathcal{I}nt$	Integer	
$\mathcal{N}$	Node	
	$\mathcal{N}o$	Other node
	$\mathcal{N}s$	Start node
	$\mathcal{N}f$	Finish node
	$\mathcal{N}b$	Begin node
	$\mathcal{N}e$	End node
$\mathcal{O}bj$	Objective	
$\mathcal{O}pt$	Option	
$\mathcal{O}s$	Objective specification	
$\mathcal{P}l$	Plan	
$\mathcal{P}$	Process	
$\mathcal{Q}$	Question	
$\mathcal{R}s$	Rationale specification	
$\mathcal{S}$	Set	
$\mathcal{S}tr$	String	
$\mathcal{T}p$	Timepoint	
$\mathcal{T}set^{\mathcal{A},\mathcal{A}}$	Timepoint pair set	
$\mathcal{U}$	Resource unit	

Appendix B

Language Definition

This section presents the language foundation for the Common Process Language (CPL). The BNF form used in this section is borrowed from the style used to describe the enhanced PIF language in "Foundations for Product Realization Process Knowledge Sharing", Knowledge Based Systems, Inc., Final Report, U.S. Dept. of Commerce, Contract No. 50-DKNB-7-90095.

B.1 BNF Conventions

- A vertical bar "|" indicates an exclusive disjunction; thus, for example, if C1 and C2 are two syntactic categories "C1|C2" indicates an occurrence of either an instance of C1 or C2 but not both. The absence of such a bar between two constructs indicates a concatenation.
- An asterisk "*" immediately following a construct indicates that there can be any finite number (including 0) of instances of the construct.
- A plus sign "+" superscript immediately following a construct indicates that there can be one or more instances of the construct.
- Braces "{" and "}" are used to indicate grouping. Thus, "{C1|C2}+" indicates one or more instances of either C1 or C2.
- A construct surrounded by brackets (e.g. "[C1|C2]") indicates that an instance of the indicated construct is optional.
- Nonterminals, representing categories of CPL expressions, start with "<" and end with ">".
- Where necessary, the space character is represented by "<space>".

B.2 Basic Tokens and Simple Expressions

```
<uc-letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<lc-letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
   <letter> ::= <uc-letter> | <lc-letter>
    <digit> ::= 0|1|2|3|4|5|6|7|8|9
```

```
  <integer> ::= <digit>+
    <float> ::= <digit>*.<digit>+
   <number> ::= {[-]<integer>}|{[-]<float>}
     <oper> ::= -|~|#|$|*|+|/
    <punct> ::= _|-|~|!|@|#|$|%|^|&|*|(|)|+|=|‘|:|;|’|<|>|,|.|?|/|||[|]|{|}|<space>
```

B.3 Base Categories of Expressions

```
    <b-con> ::= {<uc-letter>}{<letter>|<digit>}* {{_|-}{<letter>|<digit>}}*
    <b-var> ::= ?<b-con>
   <b-func> ::= {<oper>|<lc-letter>}{<letter>|<digit>}*{{_|-|.}{<letter>|<digit>}}*
   <b-pred> ::= {<lc-letter>}{<letter>|<digit>}*{{_|-|.}{<letter>|<digit>}}*
   <b-sort> ::= {<lc-letter>}{<letter>|<digit>}*{{_|-|.}{"|<letter>|<digit>}}*
<doc-string> ::= "{<letter>|<digit>|<punct>|\"|\\}*"
  <comment> ::= //{<letter>|<digit>|<punct>}* {{\n}|{\r}|{\r\n}}
```

B.4 Base Grammar for CPL

<con> ::= a member of C_λ
<var> ::= a member of V_λ
<func> ::= a member of F_λ
<pred> ::= a member of P_λ
<sort> ::= a member of S_λ
<term> ::= <atomterm> | <compterm>
<atomterm> ::= <var> | <con>
<compterm> ::= <func> (<term>{,<term>}*)
<sentence> ::= <command> | <sortdef> | <assignment> | <atomsent> |
<boolsent> | <quantsent>
<command> ::= %{define-domain|import-domain}(<pred>{,<pred>}*)
<sortdef> ::= SORT <sort> = **{** [<con> {, <con>*}]} **}**[1]
<assignment> ::= <compterm>={<con> | <doc-string> | <integer>}
<atomsent> ::= <pred> (<term>{,<term>}*)
<boolsent> ::= not (<sentence>) | and (<sentence> <sentence>+) |
or (<sentence> <sentence>+) | => (<sentence><sentence>+)
<quantsent> ::= <forall|exists> {<var> |(<var>+)} (<sentence>)

[1] Note that the bolded {} are actually terminals in this expression, so SORT cpo-action={A1,A2} is legal, and SORT cpo-action=A1,A2 is illegal.

B.5 Sample CPL Process File

This example illustrates the specification of a Purchase Brick Process which could be part of a larger house building domain such as the three pig domain described in Section 6.2.1. The graphical presentation of this process is provided on page 153.

```
%define-domain{my-building}
SORT cpo-action={A1}
SORT cpo-activity-specification={As1}
SORT cpo-begin={B1}
SORT cpo-end={E1}
SORT cpo-include-constraint={Ic1-Ic3}
SORT cpo-ordering-constraint={Or1,Or2}
SORT cpo-output-constraint={Oc1}
SORT cpo-process={P1}
SORT cpo-resource-constraint={Rc1}
SORT cpo-timepoint={Tp1-Tp4}
SORT cpo-domain-level={Dl1}

domain-level.label(Dl1)="Main Level"
domain-level.number(Dl1)=0
domain-level.contains(Dl1,P1)

label(P1)="Purchase Brick Process"
start-timepoint(P1)=Tp1
finish-timepoint(P1)=Tp4
pattern(P1)="{purchase bricks}"
label(B1)="begin"
timepoint(B1)=Tp1
include-node(Ic1)=B1
member-as(Ic1,As1)
label(E1)="end"
timepoint(E1)=TP4
include-node(Ic2)=E1
member-as(Ic2,As1)
label(A1)="purchase bricks"
begin-timepoint(A1)=Tp2
end-timepoint(A1)=Tp3
include-node(Ic3)=A1
member-as(Ic3,As1)
expression(Or1)="before(Tp1,Tp2)"
member-as(Or1,As1)
expression(Or2)="before(Tp3,Tp4)"
member-as(Or2,As1)
expression(Oc1)="{have bricks} at A1"
member-as(Oc1,As1)
expression(Rc1)="consumes {resource money} = 50 pounds at A1"
member-as(Rc1,As1)
```

Appendix C

Temporal Mapping and Analysis

C.1 Timepoint and Interval Mapping Axioms

This axiomatisation is based on Hayes' definitions [Hayes, 1996] of the isomorphic relationship between data structures in a timepoint-based theory and an interval-based theory [Allen, 1984a, Allen, 1984b].

$$(\forall a).(before(begin-timepoint(a), end-timepoint(a))) \supset timeinterval(a) \tag{C.1}$$

$$\begin{aligned}&(\forall tp_1 tp_2).(tp_1 = begin-timepoint(between(tp_1, tp_2))\wedge\\&\quad tp_2 = end-timepoint(between(tp_1, tp_2)))\\&\quad \Leftrightarrow before(tp_1, tp_2)\end{aligned} \tag{C.2}$$

$$\begin{aligned}&(\forall a_1 a_2).(timeinterval(a_1) \wedge timeinterval(a_2)\wedge\\&\quad before(end-timepoint(a_1), begin-timepoint(a_2)))\\&\quad \Leftrightarrow precedes(a_1, a_2)\end{aligned} \tag{C.3}$$

$$\begin{aligned}&(\forall a_1 a_2).(timeinterval(a_1) \wedge timeinterval(a_2)\wedge\\&\quad before(begin-timepoint(a_1), begin-timepoint(a_2))\wedge\\&\quad before(begin-timepoint(a_2), end-timepoint(a_1))\wedge\\&\quad before(end-timepoint(a_1), end-timepoint(a_w)))\\&\quad \Leftrightarrow overlaps(a_1, a_2)\end{aligned} \tag{C.4}$$

$$\begin{aligned}&(\forall a_1 a_2).(timeinterval(a_1) \wedge timeinterval(a_2)\wedge\\&\quad begin-timepoint(a_1) = begin-timepoint(a_2)\wedge\\&\quad before(end-timepoint(a_1), end-timepoint(a_2)))\\&\quad \Leftrightarrow starts(a_1, a_2)\end{aligned} \tag{C.5}$$

$$\begin{aligned}&(\forall a_1 a_2).(timeinterval(a_1) \wedge timeinterval(a_2)\wedge\\&\quad before(begin-timepoint(a_2), begin-timepoint(a_1))\wedge\\&\quad before(begin-timepoint(a_1), end-timepoint(a_1)))\wedge\\&\quad before(end-timepoint(a_1), end-timepoint(a_2)))\\&\quad \Leftrightarrow during(a_1, a_2)\end{aligned} \tag{C.6}$$

$$\begin{aligned}&(\forall a_1 a_2).(timeinterval(a_1) \wedge timeinterval(a_2)\wedge\\&\quad before(begin-timepoint(a_2), begin-timepoint(a_1))\wedge\\&\quad end-timepoint(a_1) = end-timepoint(a_2))\\&\quad \Leftrightarrow finishes(a_1, a_2)\end{aligned} \tag{C.7}$$

C.2 Results of Temporal Mapping Analysis

The following tables and figures summarize the results of the analysis procedure which was outlined on page 192. The table entries index into a graphical presentation in the figure above it. Each table/figure pairing represents a level in the space of possible timepoint-based configurations.

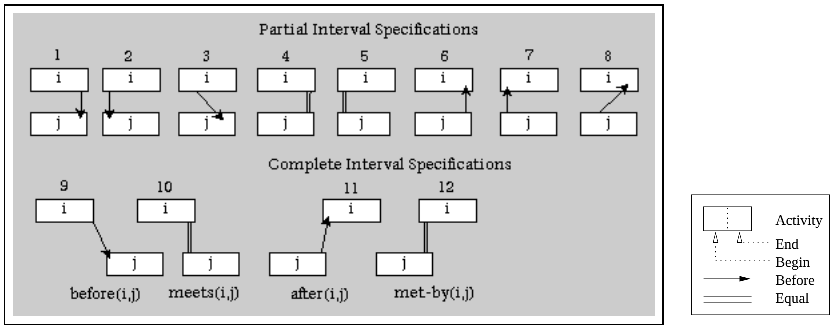

Figure C.1: Legal and unique 1-assignment specifications

C_1	
$E_1 \prec B_2$	9
$B_1 \prec E_2$	3
$B_1 \prec B_2$	2
$E_1 \prec E_2$	1
$B_1 = B_2$	5
$E_1 = E_2$	4
$B_1 = E_2$	12
$E_1 = B_2$	10
$E_2 \prec B_1$	11
$B_2 \prec E_1$	8
$B_2 \prec B_1$	7
$E_2 \prec E_1$	6

Table C.1: Possible set of 1-assignment constraint specifications

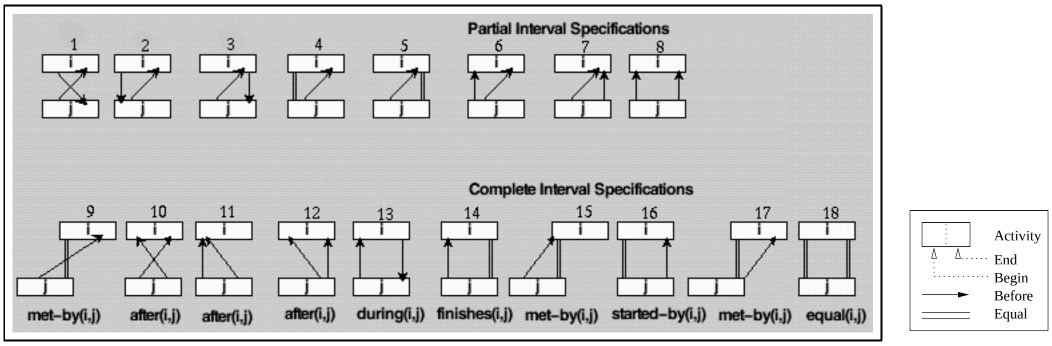

Figure C.2: Legal and unique 2-assignment specifications

$C_1/FigC.1$	11	8	7	6	5	4	12	10	9	3	2	1
$E_1 \prec B_2$	×	×	×	×	×	×	×	×	•	10	11	12
$B_1 \prec E_2$		1	2	3	4	5	9	×	10	•	6	7
$B_1 \prec B_2$			×	13	×	14	15	×	11	6	•	8
$E_1 \prec E_2$				×	16	×	17	×	12	7	8	•
$B_1 = B_2$					•	18	×	×	×	4	×	16
$E_1 = E_2$						•	×	×	×	5	14	×
$B_1 = E_2$							×	•	×	×	×	×
$E_1 = B_2$								×	×	9	15	17
$E_2 \prec B_1$									×	×	×	×
$B_2 \prec E_1$										1	2	3
$B_2 \prec B_1$											×	13
$E_2 \prec E_1$												×

Table C.2: Validity of 2-assignment constraints, ×=illegal,$< num >$=legal,•=repeat

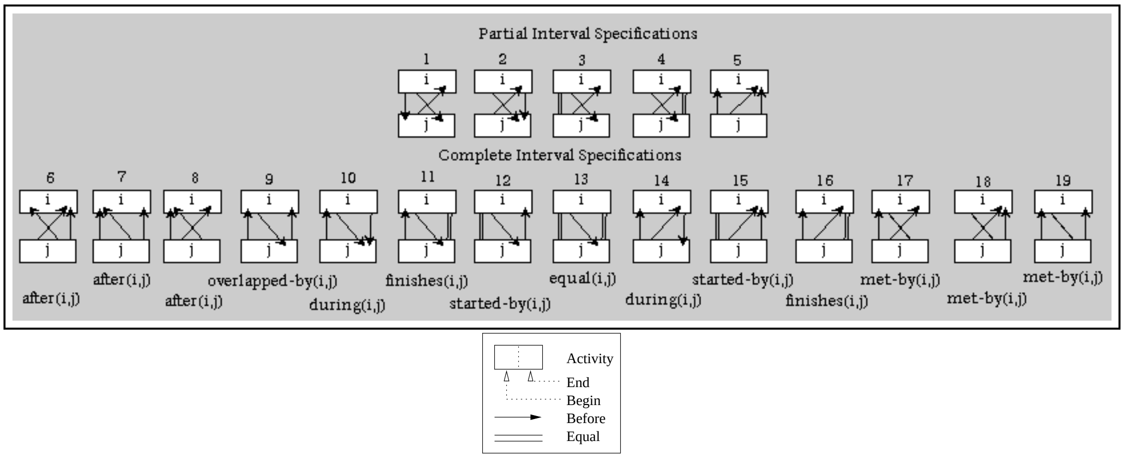

Figure C.3: Legal and unique 3-assignment specifications

$C_1/FigC.2$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$E_1 \prec B_2$	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
$B_1 \prec E_2$	•	1	2	3	4	1	2	9	×	×	×	×	10	11	×	12	×	13
$B_1 \prec B_2$	1	•	9	×	11	×	10	×	×	×	×	×	×	×	×	×	×	×
$E_1 \prec E_2$	•	9	×	12	×	14	×	×	×	×	×	×	•	×	×	×	×	×
$B_1 = B_2$	3	×	12	•	13	×	15	×	×	×	×	×	×	×	×	•	×	•
$E_1 = E_2$	4	11	×	13	•	16	×	×	×	×	×	×	×	•	×	×	×	•
$B_1 = E_2$	×	×	×	×	×	17	18	19	•	×	×	×	×	×	•	×	•	×
$E_1 = B_2$	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
$E_2 \prec B_1$	×	×	×	×	×	8	6	7	×	•	•	•	×	×	×	×	×	×
$B_2 \prec E_1$	•	•	•	•	•	•	•	9	•	•	8	6	14	16	17	15	18	13
$B_2 \prec B_1$	1	×	14	×	16	•	5	•	17	8	•	7	•	•	•	×	19	×
$E_2 \prec E_1$	2	10	×	15	×	5	•	•	18	6	7	•	×	×	19	•	•	×

Table C.3: Validity of 3-assignment constraints, ×=illegal,$<num>$=legal,•=repeat

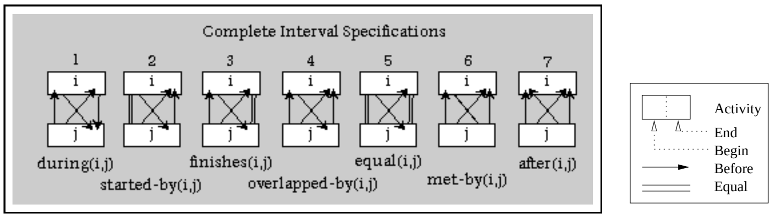

Figure C.4: Legal and unique 4-assignment specifications

$C_1/FigC.3$	1	2	3	4	9	10	11	12	13	14	15	16	17	18	19	8	6	5	7
$E_1 \prec B_2$	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
$B_1 \prec E_2$	•	•	•	•	•	•	•	•	•	1	2	3	×	×	×	×	×	4	×
$B_1 \prec B_2$	•	4	×	3	•	•	•	×	×	×	×	×	×	×	×	×	×	×	×
$E_1 \prec E_2$	4	•	2	×	×	•	×	•	×	•	×	×	×	×	×	×	×	×	×
$B_1 = B_2$	×	2	•	5	×	×	×	•	•	×	•	×	×	×	×	×	×	×	×
$E_1 = E_2$	3	×	5	•	×	×	•	×	•	×	×	•	×	×	×	×	×	×	×
$B_1 = E_2$	×	×	×	×	×	×	×	×	×	×	×	×	•	•	•	×	×	6	×
$E_1 = B_2$	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
$E_2 \prec B_1$	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	•	•	7	•
$B_2 \prec E_1$	•	•	•	•	•	1	3	2	•	•	•	•	•	•	6	•	•	•	7
$B_2 \prec B_1$	×	1	×	3	•	•	•	×	×	•	×	•	•	6	•	•	7	•	•
$E_2 \prec E_1$	1	•	2	×	•	•	×	•	×	×	•	×	6	•	•	7	•	•	•

Table C.4: Validity of 4-assignment constraints, ×=illegal,$<num>$=legal,•=repeat

Appendix D

Example CPM Work Products

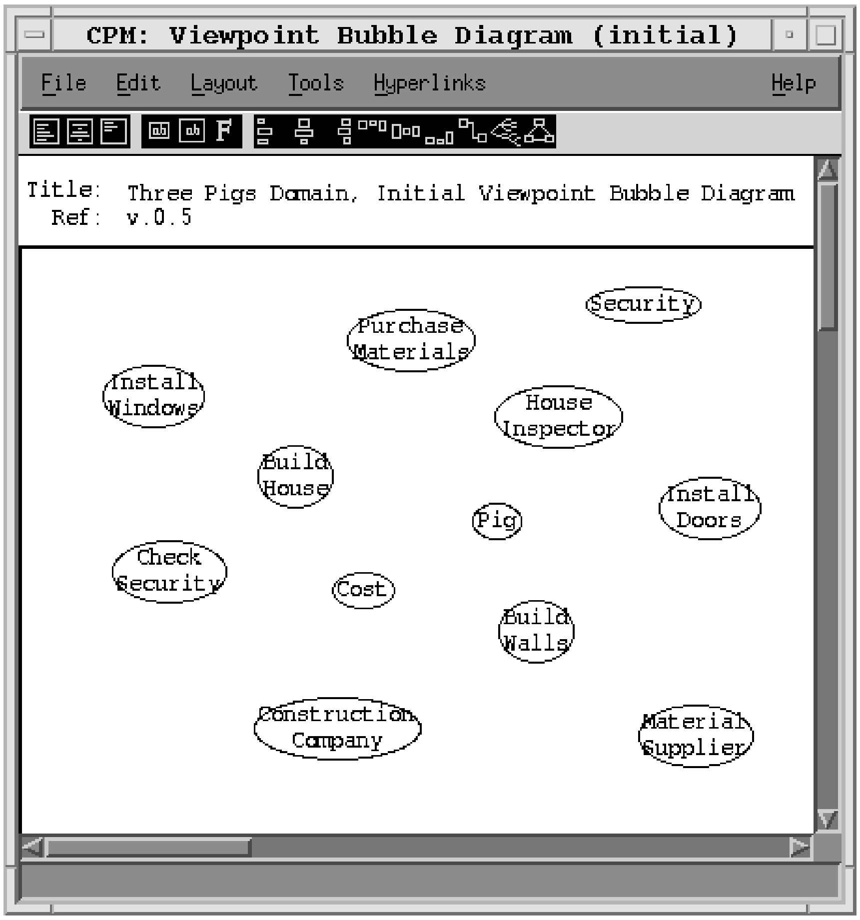

Figure D.1: Initial viewpoint bubble diagram

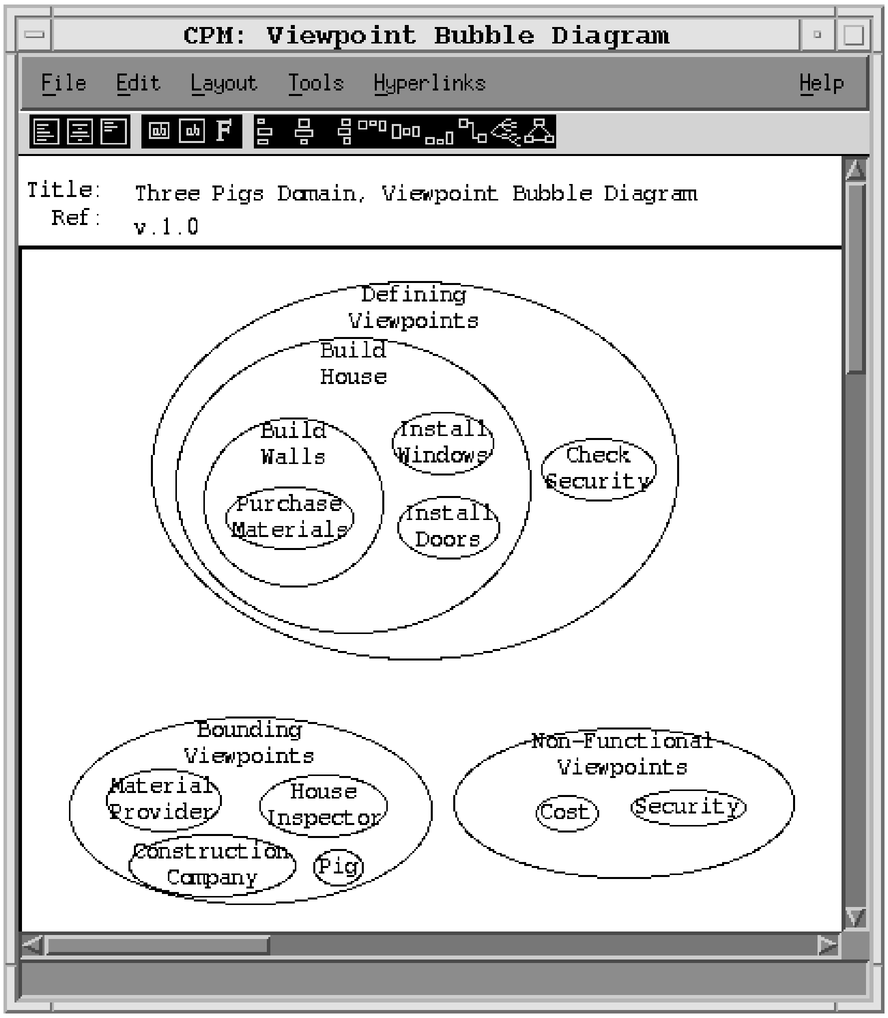

Figure D.2: Partitioned viewpoint bubble diagram

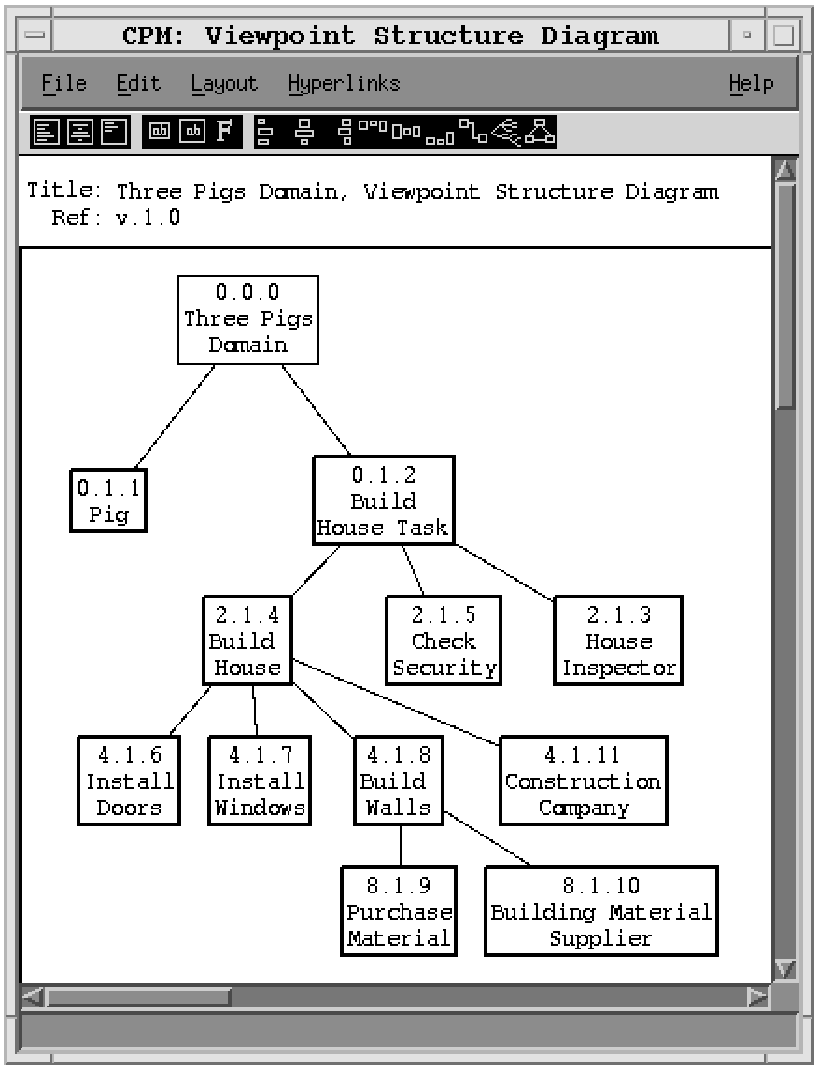

Figure D.3: Viewpoint structure diagram

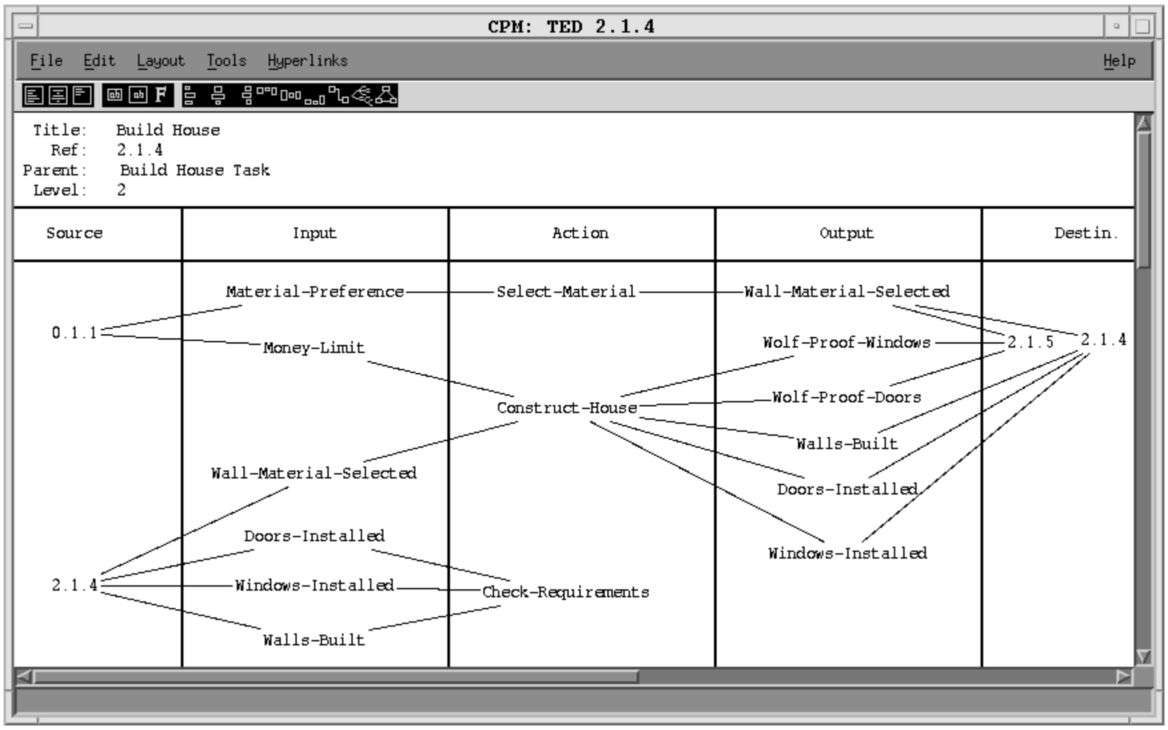

Figure D.4: Customer tabular entry diagram

Appendix E

Microwave T/R Plan

This entry provides an example process plan which roughly corresponds to those used in manufacturing a microwave transmit/relay module at Northrup Grumman. This example was initially developed for the EDAPS process planning module [Smith *et al.*, 1996, Smith, 1997]. For more information on transmit/relay modules see [Sander, 1987].

```
All length units are inches unless otherwise stated.
All time units are minutes unless otherwise stated.

Processes:

Opn A   BC/WW   Setup   Runtime   LN    Description
001 A    VMC1    2.00      0.00   01    Orient board
                                  02    Clamp board
                                  03    Establish datum point at
                                        bullseye (0.25, 1.00)

001 B    VMC1    0.10      0.43   01    Install 0.30-diameter drill bit
                                  02    Rough drill at (1.25, -0.50)
                                        to depth 1.00
                                  03    Finish drill at (1.25, -0.50)
                                        to depth 1.00

001 C    VMC1    0.10      0.77   01    Install 0.20-diameter drill bit
                                  02    Rough drill at (0.00, 4.88)
                                        to depth 1.00
                                  03    Finish drill at (0.00, 4.88)
                                        to depth 1.00
                                  04    Rough drill at (5.50, 4.88)
                                        to depth 1.00
                                  05    Finish drill at (5.50, 4.88)
                                        to depth 1.00
                                  06    Rough drill at (2.75, 1.25)
                                        to depth 1.00
                                  07    Finish drill at (2.75, 1.25)
                                        to depth 1.00
                                  08    Rough drill at (3.25, 0.00)
                                        to depth 1.00
                                  09    Finish drill at (3.25, 0.00)
                                        to depth 1.00
```

```
001 T    VMC1    2.20      1.20   01    Total time on VMC1


Opn A   BC/WW   Setup   Runtime   LN    Description
002 A     PC1    0.00     10.00   01    Condition board (deburr and degrease)

002 B     PC1    0.00     10.00   01    Pickle board

002 C     PC1    0.00     10.00   01    Activate board (sensitize and seed)

002 D     PC1    0.00     15.00   01    Electroless copper deposition

002 T     PC1    0.00     45.00   01    Total time on PC1


Opn A   BC/WW   Setup   Runtime   LN    Description
003 A     PC2   30.00      0.50   01    Setup
                                  02    Apply plating resist

003 B     PC2    0.00     25.00   01    Copper electroplating

003 C     PC2    0.00     15.00   01    Tin/lead electroplating

003 D     PC2    0.00      5.00   01    Strip plating resist

003 T     PC2   30.00     45.50   01    Total time on PC2


Opn A   BC/WW   Setup   Runtime   LN    Description
004 A     EC1    0.00     32.29   01    Pre-clean board (scrub and wash)
                                  02    Dry board in oven at 85 deg. F

004 B     EC1   30.00      0.40   01    Setup
                                  02    Spindle photoresist
                                        from heated mylar

004 C     EC1   30.00      2.20   01    Setup
                                  02    Photolithography of photoresist
                                        using phototool in "real.iges"

004 D     EC1   30.00     28.00   01    Setup
                                  02    Etching of copper

004 T     EC1   90.00     62.89   01    Total time on EC1


Opn A   BC/WW   Setup   Runtime   LN    Description
005 A    VMC1    2.00      0.00   01    Orient board
                                  02    Clamp board
                                  03    Establish datum point at
                                        bullseye (0.25, 1.00)

005 B    VMC1    0.10      0.34   01    Install 0.15-diameter side-milling tool
                                  02    Rough side-mill pocket at (-0.25, 1.25)
                                        length 0.40, width 0.30, depth 0.50
                                  03    Finish side-mill pocket at (-0.25, 1.25)
                                        length 0.40, width 0.30, depth 0.50
                                  04    Rough side-mill pocket at (-0.25, 3.00)
```

					length 0.40, width 0.30, depth 0.50
				05	Finish side-mill pocket at (-0.25, 3.00) length 0.40, width 0.30, depth 0.50
005 C	VMC1	0.10	1.54	01	Install 0.08-diameter end-milling tool
				02	Rough end-mill pocket at (0.95, 3.20) length 0.30, width 0.40, depth 0.50
				03	Rough end-mill pocket at (1.35, 3.20) length 0.30, width 0.40, depth 0.50
005 D	VMC1	0.10	1.24	01	Install 0.08-diameter slot-milling tool
				02	Rough slot-mill pocket at (2.85, 0.00) length 0.50, width 0.75, depth 0.50
005 E	VMC1	0.10	1.54	01	Install 0.40-diameter drill bit
				02	Rough drill at (0.25, -0.75) to depth 1.00
				03	Rough drill at (0.25, 4.75) to depth 1.00
				04	Rough drill at (3.00, -0.75) to depth 1.00
				05	Rough drill at (3.00, 4.75) to depth 1.00
005 F	VMC1	0.10	0.22	01	Install 0.15-diameter drill bit
				02	Rough drill at (-0.05, 1.40) to depth 1.00
				03	Finish drill at (-0.05, 1.40) to depth 1.00
				04	Rough drill at (-0.05, 3.15) to depth 1.00
				05	Finish drill at (-0.05, 3.15) to depth 1.00
005 T	VMC1	2.50	4.87	01	Total time on VMC1

Opn A	BC/WW	Setup	Runtime	LN	Description
006 A	MC1	30.00	5.71	01	Setup
				02	Prepare board for soldering
006 B	MC1	30.00	0.29	01	Setup
				02	Screenprint solder stop on board
006 C	MC1	30.00	7.50	01	Setup
				02	Deposit solder paste at (3.35,1.23) on board using nozzle
				03	Deposit solder paste at (3.25,1.23) on board using nozzle
				04	Deposit solder paste at (2.58,0.10) on board using nozzle
				05	Deposit solder paste at (2.58,0.00) on board using nozzle
				06	Deposit solder paste at (3.00,0.80) on board using nozzle
				07	Deposit solder paste at (2.90,0.80) on board using nozzle
				08	Deposit solder paste at (2.55,0.10)

```
                                              on board using nozzle
                                        09    Deposit solder paste at (2.55,0.00)
                                              on board using nozzle
                                        10    Deposit solder paste at (1.75,3.50)
                                              on board using nozzle
                                        11    Deposit solder paste at (1.55,3.20)
                                              on board using nozzle
                                        12    Deposit solder paste at (0.95,3.50)
                                              on board using nozzle
                                        13    Deposit solder paste at (1.15,3.20)
                                              on board using nozzle
                                        14    Deposit solder paste at (3.15,1.23)
                                              on board using nozzle
                                        15    Deposit solder paste at (3.05,1.23)
                                              on board using nozzle
                                        16    Deposit solder paste at (3.65,0.55)
                                              on board using nozzle
                                        17    Deposit solder paste at (3.45,0.55)
                                              on board using nozzle
                                        18    Deposit solder paste at (3.15,0.80)
                                              on board using nozzle
                                        19    Deposit solder paste at (3.05,0.80)
                                              on board using nozzle
                                        20    Deposit solder paste at (3.52,-0.20)
                                              on board using nozzle
                                        21    Deposit solder paste at (3.52,-0.50)
                                              on board using nozzle
                                        22    Deposit solder paste at (1.70,0.65)
                                              on board using nozzle
                                        23    Deposit solder paste at (1.50,0.65)
                                              on board using nozzle
                                        24    Deposit solder paste at (1.17,0.55)
                                              on board using nozzle
                                        25    Deposit solder paste at (1.17,0.25)
                                              on board using nozzle
                                        26    Deposit solder paste at (-0.05,2.40)
                                              on board using nozzle
                                        27    Deposit solder paste at (-0.25,2.40)
                                              on board using nozzle
                                        28    Deposit solder paste at (0.32,4.30)
                                              on board using nozzle
                                        29    Deposit solder paste at (0.32,4.00)
                                              on board using nozzle
                                        30    Deposit solder paste at (3.52,4.30)
                                              on board using nozzle
                                        31    Deposit solder paste at (3.52,4.00)
                                              on board using nozzle

006 D     MC1    0.00      5.71     01    Dry board in oven at 85 deg. F
                                              to solidify solder paste

006 T     MC1   90.00     19.21     01    Total time on MC1


Opn A   BC/WW   Setup   Runtime     LN    Description
007 A     AC1   30.00      1.50     01    Setup
                                        02    Put adhesive on board at (0.95,3.20)
                                              for component D1
```

```
                                 03    Put adhesive on board at (1.35,3.20)
                                       for component D2
                                 04    Put adhesive on board at (2.85,0.00)
                                       for component FET

007 B     AC1   30.00     0.00   01    Setup

007 C     AC1    0.00    10.00   01    Wait 90-120 minutes for the furnace
                                       to come to profile, then cure adhesive
                                       on board at 125-150 degrees C

007 T     AC1   60.00    11.50   01    Total time on AC1


Opn A   BC/WW   Setup  Runtime   LN    Description
008 A     MC1    0.00     7.50   01    Pick-and-place C1 at (3.35,4.00)
                                 02    Pick-and-place C2 at (0.15,4.00)
                                 03    Pick-and-place C3 at (-0.25,2.25)
                                 04    Pick-and-place C4 at (1.00,0.25)
                                 05    Pick-and-place C5 at (1.50,0.50)
                                 06    Pick-and-place Cs at (3.35,-0.50)
                                 07    Pick-and-place Cf at (3.05,0.75)
                                 08    Pick-and-place Cc at (3.45,0.40)
                                 09    Pick-and-place Cd at (3.05,1.20)
                                 10    Pick-and-place D1 at (0.95,3.20)
                                 11    Pick-and-place D2 at (1.35,3.20)
                                 12    Pick-and-place Rg at (2.50,0.00)
                                 13    Pick-and-place Rf at (2.90,0.75)
                                 14    Pick-and-place Rs at (2.50,0.00)
                                 15    Pick-and-place Rd at (3.25,1.20)

008 B     MC1   30.00     5.00   01    Setup
                                 02    Wait 90-120 minutes for the furnace
                                       to come to profile, then reflow solder
                                       with hot air at 205 degrees C

008 C     MC1   15.00     0.00   01    Setup for hand soldering

008 D     MC1    0.00     7.00   01    Hand solder L1 at (1.00,-0.15)
                                 02    Hand solder L2 at (1.20,0.50)
                                 03    Hand solder FET at (2.85,0.00)

008 E     MC1    0.00    11.43   01    Spray board with vapor to
                                       clean flux

008 T     MC1   45.00    30.93   01    Total time on MC1


Opn A   BC/WW   Setup  Runtime   LN    Description
009 A     TC1    0.00    35.00   01    Perform pre-cap testing on board

009 T     TC1    0.00    35.00   01    Total time on TC1


Opn A   BC/WW   Setup  Runtime   LN    Description
010 A    HSC1    0.00     1.86   01    Hermetically seal board

010 T    HSC1    0.00     1.86   01    Total time on HSC1
```

Opn	A	BC/WW	Setup	Runtime	LN	Description
011	A	TC1	0.00	35.00	01	Perform post-cap testing on board
011	B	TC1	0.00	29.67	01	Perform final inspection of board
011	T	TC1	0.00	64.67	01	Total time on TC1
999	T		319.70	322.63	01	Total time to manufacture

Appendix F

Task Formalism Workstation