A Resource ontology for Enterprise Modelling

Masters of Applied Science 1994

Fadi George Fadel

Department of Industrial Engineering

University of Toronto

ABSTRACT

The role of computing has become an essential function in any enterprise hoping to achieve a competitive advantage. How effectively the enterprise functions does not depend only on labour, management and capital, but also on the availability of information needed for decision making. Accordingly, information systems are playing an increasingly central role in the day to day operations of organizations. A critical component of an information system is the data model used to represent the organization: its activities, resources, organization units, people, etc. With the advent of distributed information systems, the availability of a shared model that enables different units of the enterprise (human or machine) to understand each other has also increased in importance.

The research focuses on defining a generic and well defined set of terms required for modelling resources and required in integrating enterprise functions. The need for a resource ontology arises out of activities such as planning, scheduling (etc.) whose fundamental decision processes revolve around the allocation resources.

Acknowledgment

My thanks to everyone who contributed to the presented thesis, in form of discussions and proof reading efforts. I am specially grateful to Professor Mark S. Fox for his most valuable and helpful supervision. Furthermore I want to express my appreciation for putting up with me in my graduate studies period and his discussions regarding the research. I express my appreciation for Professor Chelliah Sriskandarajah for his time and valuable comments. My gratitude is also due to Professor Joseph C. Paradi for accepting to be member on the examination committee.

I wish also to express my thanks to Henry Kim, Hyame Fadel, John Blake, John Chionglo, K. Donald Tham, Michael Grunninger, Roula Haddad and Ziad Haddad for their effort in proof reading the different chapters of the thesis.

Furthermore, I am also thankful for all members of the Enterprise Engineering Laboratory for their support and beneficial comments.

Last but not least, my thanks are also due to my family for their valuable moral support especially during time of frustration.

This research is supported, in part, by the Natural Science and Engineering Research Council, Digital Equipment Corp., Micro Electronics and Computer Research Corp., and Spar Aerospace.

To my parents: George and Josephine Fadel

Table of Contents

CHAPTER 3

CHAPTER 4

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 *Introduction*

For an enterprise to remain competitive computing must be seen as an essential function. The effectiveness of an enterprise depends not only on labour, management and capital, but also on the availability of information needed for decision making actions that can be shared among different applications. Throughout the coming chapters the ontology for modelling one of the most basic enterprise components, resources, is presented.

1.1 Introduction

"It has become apparent that a competitive and efficient enterprise does not solely depend on capital and management, but also depends on the accessibility of information and the ability to coordinate both decisions and actions" [Fox 1992]. Organizations are striving towards efficient collaboration between organizational units, however this effort is often hampered especially in large organizations. The problem arises because of the inability of organizational units to share information and coordinate actions. Efficient communication among different organizational enterprise units exists if and only if there exists a common understanding what is being communicated. This research is concerned with defining a generic representation of one of the most basic components of an enterprise - a resource. The *enterprise resource ontology* focuses on defining a generic set of terms, in machine and human readable form, required for modelling resources. This enables the interaction of different enterprise units by providing a standard common language needed for the representation of information about a specific domain of an enterprise.

The goal of the research is to create a *resource ontology* for use in manufacturing enterprises. An ontology is defined as a set of terms and meanings that enable two agents to exchange knowledge for some purposes in some domain [Tenenbaum 92]. An ontology is comprised of data model composed of objects, attributes and relations and the formal definitions of constraints and terms in the data model, in first order logic. Resource ontology is defined by defining:

1. The span of the model, by a set of competency questions[1].
2. A sharable ontology.
3. The definition of each term in the ontology First Order Logic (FOL).
4. An implementation of the definitions as axioms in Prolog.

Our approach to developing ontologies has several benefits:

1. The range of application supported by the ontology are defined clearly by competency questions.
2. The semantics of each term in the ontology is clearly defined in FOL.
3. Common sense reasoning is supported by implementing the definitions in prolog and using the axioms for deductive query processing.

1.2 The need for resource modelling framework

Knowledge based systems are currently described as being *isolated monoliths*[2]. These systems are characterized by having domain dependent terminology, facts, axioms etc. [Gruber 1991]. Because of this, these monolithic systems are constrained by an inability to inter-communicate which leads to the inability to reuse or share the domain knowledge base and tools. This has also led to different representations of the same knowledge defined with inadequate semantic definitions, which results in inconsistent and different interpretations of the same knowledge. Furthermore, these representations are passive[3] and do not support reasoning at various levels of abstraction. This has made it difficult to integrate different and independent software. The key to integrating different functional software and sharing domain knowledge and tools, is to have the knowledge expressed with minimum assumptions of the application.

All enterprises are activity oriented to achieve a number of business objectives. These objectives can be fulfilled only if the needed resources are available. Consider the case where a scheduling software/agent tries to communicate with an inventory management software/agent about the availability of a resource. It is evident that a clear and common understanding should exist between the two agents about the nature of the specified resource. Similarly, among other enterprise func-

[1] Defined later

[2] That is true for all software systems

[3] "For example, if the representation contains a `works-for' relation and it is explicitly represented that Joe `works-for' Fred, and that Fred `works-for' John, then the obvious deduction that Joe `works-for' John (indirectly) cannot be made within the representation system" [Fox 92].

tions such as engineering, purchasing, accounting and sales, there should be a common understanding of resources. Therefore, the aim of the resource ontology is to act as a coupling between different enterprise functions and their respective knowledge and tools. In other words, the ontology acts as knowledge-level protocol for input, output and communication [Fox & Tenenbaum 91] [Gruber 91].

1.3 Evaluation/Design Criteria for ontology

Though there exists a number of efforts seeking to create sharable representation of enterprise knowledge - CIM-OSA [Esprit 90], ICAM [Martin 83], IWI [Scheer 89], PERA [Williams 91]- little comparative analysis has been performed. Recently, two sets of evaluation criteria have been proposed. Fox & Tenenbaum have proposed the following criteria as a basis for evaluating an ontology:

- **Generality:** The representation should be as general as possible so that it could be shared by different activities and be used by different applications.
- **Competence:** How well does the model support problem solving?
- **Consistency:** Consistency of the approach towards the enterprise activities (Uniformity of approach).
- **Perspicuity:** The model should have an adequate, understandable and simple descriptive methodology.
- **Granularity:** Can the representation be used for supporting reasoning at different levels of abstraction?
- **Transferability:** Can the model be transformed to be used for a specific application (Industry)?
- **Efficiency:** Measured in terms of space and inference.
- **Scalability:** Can the model be used for large scale applications?
- **Extensibility:** The model should have the ability to be extended without major changes.

The *competency* criterion has been chosen to evaluate the ontology. Competency questions define the types of queries available to support a task. The creation of the competency questions and the ontology is an iterative process; the competency questions drive the ontology which in turn results in the modification of the competency questions.

The obvious way to demonstrate competence is to show how a set of questions can be answered by the ontology. If no inference capability is to be assumed, then question answering is strictly reducible to "looking up" an answer that is represented explicitly. In defining a shared representation, a key question then becomes: should we be restricted to just a terminology? Should the terminology assume an inheritance mechanism? Or should we assume that some type of theorem proving capability is provided, say, in a logic programming language with axioms restricted to Horne clauses (i.e., Prolog)? What is the *deductive capability* that is to be assumed by an ontology? We propose that for each category of knowledge, a set of questions be defined that the ontology can answer. Given a representation and an accompanying theorem prover (perhaps Prolog), questions can be posed in the form of queries to be answered by the theorem prover. Given that a theorem prover is the deduction mechanism used to answer questions, the *efficiency* of a representation can be defined by the number of LIPS (Logical Inferences Per Second) required to answer a query [Fox 92].

As discussed, the competence criterion focuses on how well the model supports various tasks. We define a model's level of competence by a set of questions it should be able to answer either directly or through deduction. Following are a subset of questions we have considered in the creation of the **TOVE** model.

- **Quantity:** How much of the resource exists at time t?
- **Consumption:** Is the resource consumed by the activity? If so, how much?
- **Divisibility:** Can the resource be divided and still be usable? Can two or more activities use the resource at the same time?
- **Structure:** What are the subparts of resource R?
- **Capacity:** Can the resource be shared with other activities?
- **Location:** Where is resource R?
- **Commitment:** What activities is the resource committed to at time *t*?
- **Trend:** What is the capacity trend of a resource based on the machine usage history?

1.4 TOVE testbed

This research is being undertaken as part of the ***TOVE***[1] project The TOVE Project includes two major undertakings: the development of an Enterprise Ontology, and a testbed. The TOVE Enterprise Ontology provides a generic, reusable ontology for modelling enterprises. The TOVE ontology currently spans knowledge of

[1] TOronto Virtual Enterprise

activity, state, time, causality, resources, cost and quality. The ontology's data model is implemented on top of C++ using the Carnegie Group's ROCK knowledge representation tool and the axioms are implemented in Quintus Prolog.

The TOVE Testbed provides an environment for analyzing enterprise ontologies. The Testbed provides a model of an enterprise - a lamp manufacturing plant -, and tools for browsing, visualization, simulation and deductive queries.

1.5 Overview of the approach in this document

In this document, resource related terms that have already been defined in previous research will be merged with some new terms to reach a generic reusable ontology describing a resource. The terms are formally and rigorously defined in the form of First Order Logic axioms.

The flow in this document is as follows:

- chapter two presents a review of related research.

  Three reference models are reviewed: IWI, CIM-OSA, Purdue. Moreover five endeavors from AI domain are also reviewed.
- chapter three contains an overview of the activity-state ontology defined in TOVE.
- chapter four presents the ontology of resources. This chapter presents the semantic requirement for modelling resources.
- chapter five presents the capacity recognition process incorporated with the resource ontology.
- chapter six presents the competency questions.
- chapter seven contains the conclusion.

CHAPTER 2 Related Research

In the literature, there exist a number of attempts to define an enterprise model that provides an integrated modelling framework. In this chapter, an overview of three models - IWI, CIM-OSA and PERA - is presented. Some endeavors from the AI discipline are also presented: SIPE, KRSL, OPIS, Gerry and Cyc.

2.1 Introduction

Defining a manufacturing data model, with a definition of each object and its corresponding attributes, has been the focus of a number of efforts since the 1960's. IBM's COPIC's Manufacturing Resource Planning system (MRP), for example, has a shared database with an integrating data model. Recently, there has been efforts for providing more generic data model and modelling framework of enterprise knowledge. These efforts claim also to provide a sharable and reusable model of enterprise knowledge. From the surveyed efforts, three models are presented in this chapter because they are the most advanced and complete models available. The reviewed enterprise reference models are:

- IWI [Scheer 89]
- CIM-OSA [ESPRIT 91a]
- Purdue Enterprise Reference Architecture [PERA 91]

Furthermore, efforts form the AI literature are also reviewed: SIPE [Wilkins 88], KRSL [Allen et al 92], OPIS [Smith 90], Gerry [Zweben et al 93] and Cyc [Guha et al 90] projects. Some of the efforts aim to present generic ontology for planning and scheduling purposes.

The focus of this chapter will be directed towards identifying how resources are modelled in these systems.

2.2 Data Modelling of an Enterprise - IWI[1]

The IWI data model is function-specific data model developed in the Institue für Wirtschinformatik, Universitat des Saarlandes. The IWI data model presents a data dictionary of objects and their relationships that are required for modelling functions in a manufacturing enterprise. IWI presents a data processing solutions/models of different enterprise functions such as: production, engineering, purchasing, sales, personnel, accounting, administration etc.

Resource, functions, time etc. are represented as data objects. For example the function **Operation Assignment** is modelled in IWI context as a relation between **Operation** and **Equipment Group** entities. In other words "operation assignment" assigns an operation/activity to a resource (Equipment group). In this case, the "operation assignment" function uses a certain resource denoted by equipment group number. Accordingly the operation assignment have will have an attribute denoting to the equipment group.

Figure 1 presents the context in which an entity is considered as a resource. It shows the an "equipment group" is a resource. "Tool", "personnel" and "equipment" are other entities that are considered as resources.

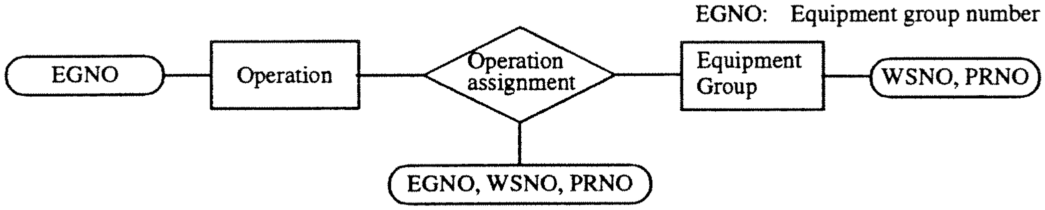
FIGURE 1 Operation assignment function in Scheer

Relational model: Operation assignment

Operation Assignment (PNO, Date, WSNO, PRNO, EGNO, ...)

Legend

PNO:	Part number
WSNO:	Work schedule number
PRNO:	Process number
EGNO:	Equipment group number



Part and *structure* constitute the basic data structure for bills of material. A structure is an object that consists more than one object (part). A part (component) in a structure could also be a structure. The data structure for both part and structure entities are presented in figure 3.

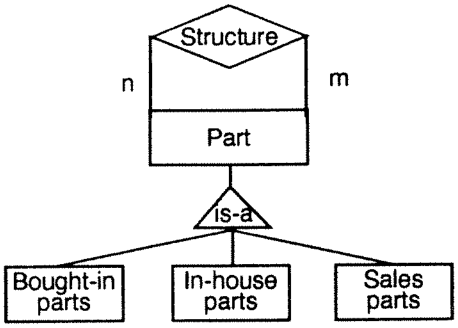
There are three sub-types of part:

- **bought-in** parts are purchased material and assemblies

[1] [Scheer 89]

- **in-house** parts are parts that involve at least one production or assembly operation
- **sales** parts consists of saleable parts: end products, spare parts

FIGURE 2 ERM of part, structure and part sub-types



As mentioned the IWI model presents typical attributes needed for part and structure description. Part is described through defining a number of attributes that specify identification, classification, status, cost data, technical-physical variables and cost data. Similarly, IWI defines the needed attributes for structures and subtype definition (figure 3). The defined attributes address a number of resource perspectives. Classification attributes are defined specifying part type, part value, requirement and planning data. Status data specify the utilization and production state of a part. Technical-physical variables are also defined specifying technical/ physical dimensions, unit of measure, quantity of a resource. Finally, different cost data are defined.

FIGURE 3 General attributes for *Part* and *Structure* entities

<u>For *part* entity</u>

Identification Data:
Part number:
Drawing number:
Number is sales catalog:
Part description:

Classification Data:
Part type:
Value:
Planning level:
Requirement type:
Planning type:

Status Data:
Production state:
Utilization state:
Amendment date:

Technical-physical variables:
Dimension:
Color:
Weight:
Space-requirement:
Measurement unit:
Inventory level:

Planning Data:
Planning form:

Cost Data:
Amendment date:
Batch costs:
Inventory units cost rate:
Cost accounting results:

<u>For *sub-types* of part entity</u>

Type: Bought-in part:
Ordering policy:
Limiting values:
Safety factor:

Type: In-house part:
Lot size:
Throughput time:
Work schedule:
Requirement type:
Planning type:

Type: Sales part:
Sales price:
Discount classification:
Minimal packing sizes:

<u>For *structure* entity</u>

Quantity:
Reject rate:
Lead time:
Operation assignment:
Date of validity
Date of phase out:
Type of bill of materials:
Variant information:

Similar to "part" and "structure", the IWI model defines typical data attributes for defining "equipment group", "equipment", "tool" and "personnel" entities as presented in figure 4. Equipments, tools, personnel and parts/structures are the data objects that are considered as resources. "Equipment group" specifies an aggregation of equipment. The attributes defined for the "equipment group" specify information such as waiting time, set-up time, load level of the group. "Equipment" entity contains attributes specifying the amendment date, maintenance policy, and the equipment group to which the equipment belongs to. Finally, the "personnel" entity contains attributes specifying performance evaluation policy, appointment date, qualification, title and the overtime payment.

In "equipment group" and "tool" entities contain attributes for defining capacity. In "tool" entity "capacity in assembly hours" attribute is used for defining the capacity of the tool. Moreover the capacity of an "equipment group" is also described in hours. However, no attributes indicate which operation(s) (activities) can use the equipment group nor specify which equipment, in the equipment group, that can support more than one operation.

FIGURE 4 Attributes for "Tool", Personnel", "Equipment" and "Equipment group" entities

For *equipment group* entity

Equipment Group Number [EGNO]:
Cost Center:
Type of workplace:
Location:
Foreman:
Planning data:
Number of working days per week:
Number of shifts per working day:
Maximum number of shifts per working day:
Number of hours per shift:
Maximum hours overtime:
Utilization level:
Performance time rate:
Waiting time for the equipment group:
Set-up time:
Load levelling:
Hourly machine rate:

For *equipment* entity

Equipment Group Number [EGNO]
Equipment Number [EQNO]:
Acquisition date:
Planning date:
Period capacity of the equipment in hours:
Date of the next preventive maintenance:
Duration of the preventive maintenance:
Number if maintenance order just completed:
Depreciation date:

For *personnel* entity

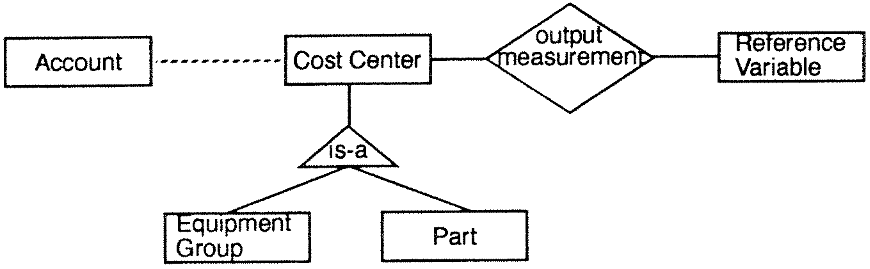
Personnel number [PERSPNO]:
Appointment date:
Previous employer:
Qualification:
Title:
Date of last assessment:
Assessment evaluation:
Date of next evaluation:
Performance evaluation:
Standard evaluation:
Overtime Payments:

For *tool* entity

TNO:
Purchase date:
Capacity in assembly hours:

As for taking account of what is produced in the factory and which activities are responsible for the act of production, the production planning and control data model has a link with the inventory accounting data model for the purpose of recording the consumption and production of parts. This is achieved through defining an account in inventory accounting that is set up for each type of material (part). As shown in figure 5, each "part" is connected to "account" entity where the addition or subtraction of any amount of the resource is recorded.

FIGURE 5 Inventory Accounting



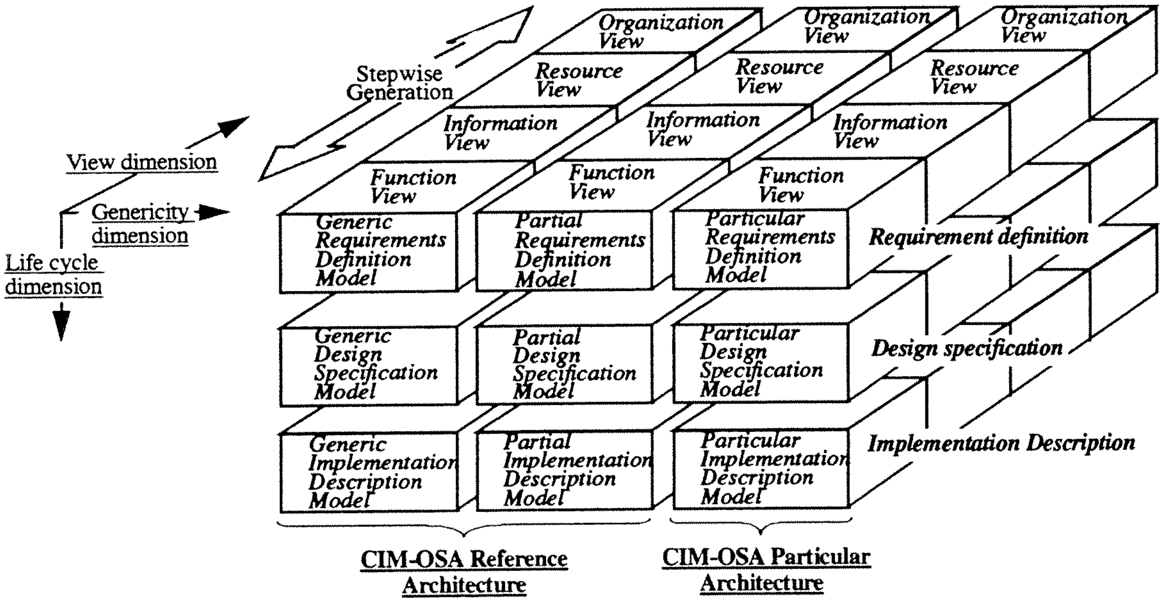
2.3 Computer Integrated Manufacturing - Open System Architecture (CIM-OSA)[1]

CIM-OSA is a project being developed by the AMICE[2] group in ESPRIT[3]. The objective of this project is to define an Open System Architecture for the manufacturing industry in general and discrete manufacturing industry in specific. The modelling methodology consists of a set of constructs (building blocks) which are to be used for structured description of business requirements and for CIM system design and implementation.

2.3.1 CIM-OSA modelling framework

The basis of the framework is a ***Reference architecture*** from which a ***Particular architecture*** is developed. The framework is represented in what is known as *CIM-OSA Cube* as shown in figure 6. The framework also provides a structure which elucidates the relations between the information technology, manufacturing technology and the applications tools which are needed to operate the enterprise.

FIGURE 6 Overview of the architecture Framework - CIM-OSA cube


Stepwise Generation
View dimension
Genericity dimension
Life cycle dimension
Organization View
Organization View
Organization View
Resource View
Resource View
Resource View
Information View
Information View
Information View
Function View
Function View
Function View
Generic Requirements Definition Model
Partial Requirements Definition Model
Particular Requirements Definition Model
Requirement definition
Generic Design Specification Model
Partial Design Specification Model
Particular Design Specification Model
Design specification
Generic Implementation Description Model
Partial Implementation Description Model
Particular Implementation Description Model
Implementation Description
CIM-OSA Reference Architecture
CIM-OSA Particular Architecture

[1] [ESPRIT 91a]

[2] AMICE: European Computer Integrated Manufacturing Architecture.

[3] ESPRIT: European Strategic Program for Research and Development in Information Technology.

2.3.2 CIM-OSA Reference Architecture

CIM-OSA reference architecture consists of a set generic building blocks, partial model and guidelines for the three modelling levels as presented in the CIM-OSA cube (figure 6).

2.3.3 Levels of modelling

Modelling levels in CIM-OSA represent the three different stages of developing an enterprise model. These levels are:

1. **Requirement Definition Modelling Level:** The level describes *what* is to be performed in order to achieve enterprise goals.
2. **Design Specification Modelling Level**: This level optimizes and structures the business requirements defined in the previous level, based on different constraints.
3. **Implementation Description Modelling Level**: states explicitly an integrated set of components (H/W, S/W) required for effective realization of enterprise operations.

2.3.4 The CIM-OSA views

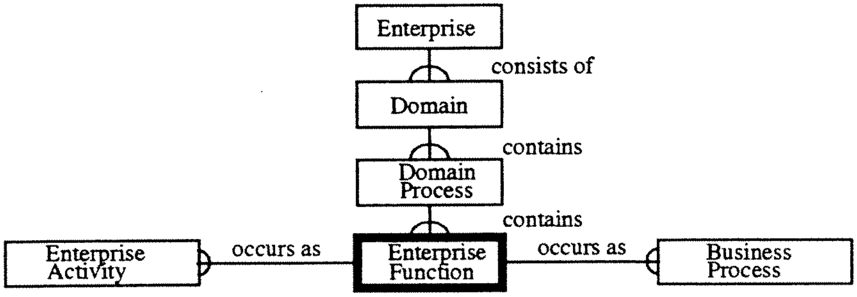
CIM-OSA has defined four views with which it is intended to model different aspects of an enterprise. These views are: function view, information view, organization view and **resource view**.

2.3.4.1 Function view

This view is used to define the structure and content of the whole enterprise or part/section of it, which is called *Domain*. Two of the most basic constructs in the function view are the *domain* and the *enterprise function* (figure 7). A *Domain* is a construct that specifies the part or parts of an enterprise required to achieve an enterprise objective. The *Enterprise Function,* represents what tasks[1] required to achieve a specific enterprise objectives.

[1] Task is a general term that points to either to Domain Process, Business process or Enterprise Activity.

FIGURE 7 Relationships in the function view



2.3.4.2 Information View

The information view defines *a system* for modelling, storing, processing information across diverse and heterogenous applications [Beeckman 90]. Similar to the function view, the information view is defined over the three different modelling levels.

2.3.4.3 Organization view

This view contains information on the different responsibilities within an enterprise. It is hierarchically structured using the concept of 'cells/groups' for grouping the organizational responsibility. At the *Requirement Definition Modelling Level*, responsibilities for manipulating human and machine decision making processes, are defined. At the *Design Specification Level*, an optimized version, of the requirements set in the requirement level, is defined. Finally the basic contribution of the organization view at the *Implementation Modelling Level* is defining the configuration of the physical system.

2.3.4.4 Resource view

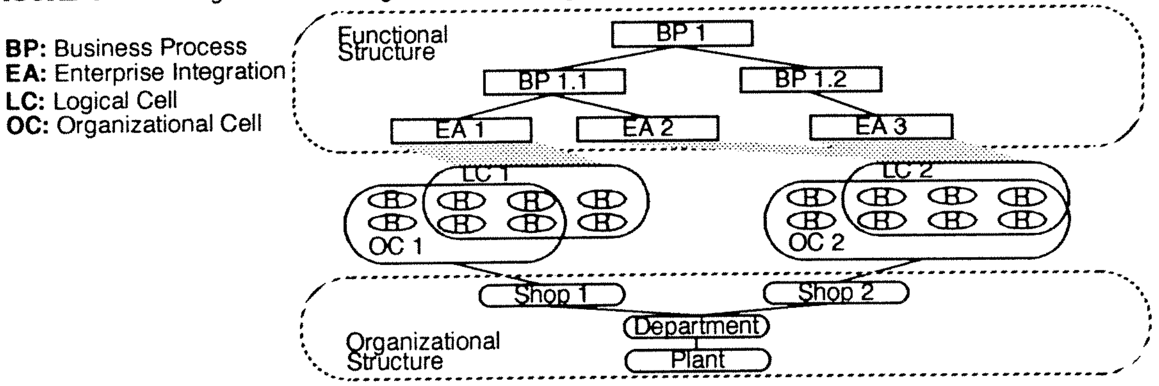
Resources are modelled in CIM-OSA's framework through the resource view which contains all the related information concerning the enterprise resources. To date no **detailed** description of constructs have yet been developed [ESPRIT 91a]. The view is hierarchically structured based on the *logical resource cell* grouping which is dependent on the requirements of the enterprise operations. The resource view is accomplished over the three modelling levels: requirement, design and implementation.

At the Requirement Definition Modelling Level: The resource view at the *Requirement Definition Modelling Level* has not been formally developed yet [ESPRIT 91a] [ESPRIT 91b]. However as previously mentioned this level provides *what* has to be done within the enterprise - in this case in terms of resource requirements.

At the Design Specification Modelling Level: A set of optimized and balanced resources, required to support a set of Enterprise Activities, are defined. The resource set is included in the ***Logical Cell*** building block. A "Logical Cell" is a building block construct in which logically structured resources are grouped. The aim of this building block is to identify a number of equipments and resources which are candidates for having a "high degree of integration because they support groups of functions which require close or frequent interaction (job-oriented/process-oriented)." [ESPRIT 91a]. That logical grouping may reflect a job-oriented structure or process-oriented one[1](figure 8).

Resources in the resource view are grouped in "logical cells". The logical cell structure of a resource - LC - collects resource objects required for the implementation of enterprise activities (EA). The resource objects could be: human beings, machines, data storage and processing capabilities etc. A similar cell is also defined in this level and that is the "Organizational Cell"[2]. The "Organizational Cell" structures the resources according to their provision or their identified responsibilities.

FIGURE 8 Logical Cell vs. Organizational Cell grouping



Two resource related constructs are defined in this level:

- **Alternative resources** construct specifies the set of resources that are candidate to support an activity.
- **Specified resources** construct is derived from the alternative resources construct by optimization and specifies which resource(s) that is to support that activity.

[1] In the organizational view there exist a similar notion - ***Organizational Cell*** - which groups the resources according to their provision or their identified responsibilities.

[2] Defined in the organization view.

- **Specified Capabilities** is a result of choosing a resource(s) in the *specified resources* construct specifying the capability of the resource. Still it is not defined how that is accomplished.

At the Implementation Description Modelling Level: At this level, the view represents the realization of the above two levels by defining the physical system/equipment which includes machines, people and application programs. In this level a description of resources is presented in terms of all the tangible components used in its CIM system. These components are presented in TABLE 1.

TABLE 1 Resource components in the Implementation Description Level

Human Resources	**Manufacturing Resources**	**Information Technology**	
Work plans	Machine programs	Enterprise Engineering Software	Enterprise Operation Software
Human skills	Control programs	Integrating infrastructure	
People	Machines	Basic data processing services	
		Data processing devices	

Two resource related constructs are defined in this level:

- **Implemented Resources** construct specifies the resources which are derived from specified resources construct defined at the design specification level. In other words it is just the assertion of which resource is chosen at the implementation level to support the activity
- **Implemented Capabilities** construct specifies the capabilities provided as a result of a certain resource choice.

Figure 9 presents the relationships between the different resource constructs defined over the four views.

FIGURE 9 Resource constructs and its relation with function, information and organization view

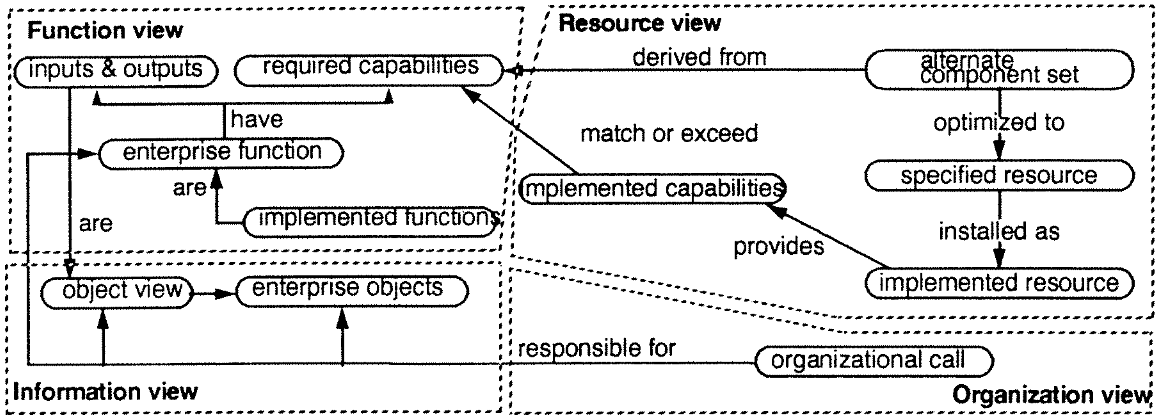


FIGURE 10 Business process definition [ESPRIT 91b]

Business Process: BP-3.7
Type: Small Batch Machining
Identifier: BP-3.7
Name: Drive Shaft Milling
Part description:
Design Authority: Manufacturing
Objective:
Constraint:
Declarative Rule:
Description:
Function Input: Drive Shaft Batch Material
Function Output: Drive Shaft Batch Material
Control Input:
Control Output:
Resource Input: Milling Machine
Resource Output: Milling Machine
Ending Status:
Comprises:
Procedural Rule

Since the resource view is not yet described in detail, there is no clear definition of what a resource is. However from the "function" and "business process" [ESPRIT 91b p. 145] definition, a reusable resource is considered as a "resource input" while a consumable one is considered as a "function input" (figure 10). Reference to "resource components in the implementation description level" table (TABLE 1) all mentioned resources are reusable ones. Machines, software, human skills are examples of such resources. Resource capabilities are modelled through the related resource constructs defined over the modelling levels. These constructs are *specified capabilities* and *implemented capabilities*. It is not clear how the capabilities are defined. An example of the definition of "human capability is as follows: **understand:** reports events, **judge:** accept failure, **decide:** start of "Update Shop Capacity". Furthermore, an example of "machine capability" is as follows: **X-length** in [200,500], **Y-length** in [100, 300], **Z-length** in [200, 350], **accuracy** < 0.005.

2.4 Purdue Enterprise Reference Architecture (PERA)

The Purdue Enterprise Reference Architecture[1] is being developed in the Laboratory for Applied Industrial Control at Purdue University. PERA was initially developed for CIM systems, but the architecture has been extended to cover modelling of any enterprise in any industry.

2.4.1 Purdue Modelling framework

The history of integrating an enterprise is covered by the so-called PERA's skeleton (figure 12). The skeleton is just a graphical description of the modelling framework which in turn is subdivided into layers/levels of modelling specifying different stages of modelling. These levels are: identification of the CIM business entity, concept layer, definition layer, specification layer, detailed design layer, manifestation layer and operations layer.

PERA also has two views of the architecture:

1. **Functional view** (definition layer): This view deals only with *how* the different tasks are performed.
2. **Implementation view**: Also addressed as the physical view and is a collection of the human organization, hardware and software needed to perform a function.

Purdue architecture contains two basics constructs: task, function. **A** ***task*** is the lowest construct that is considered when performing functional decomposition of a business entity or even an enterprise. A task is a transformation process that takes inputs and produces outputs. **A** ***function*** is a collection of tasks having a common objective. Any task and/or function is classified as either informational or manufacturing[2] tasks. An informational task takes informational inputs and produces output(s). A manufacturing task, on the other hand, takes material and/or energy inputs and transforms these into outputs. Both tasks and functions are enabled by ceratin parameters depending on which stream is being performed - informational or manufacturing.

[1] [PERA 91]

[2] Also addressed as either a physical or a customer service task

FIGURE 11 Definition of the Task Module

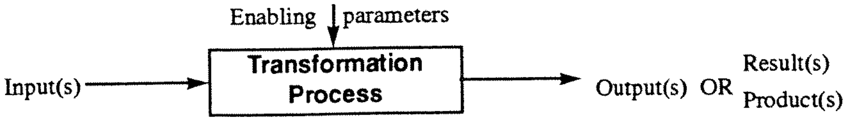
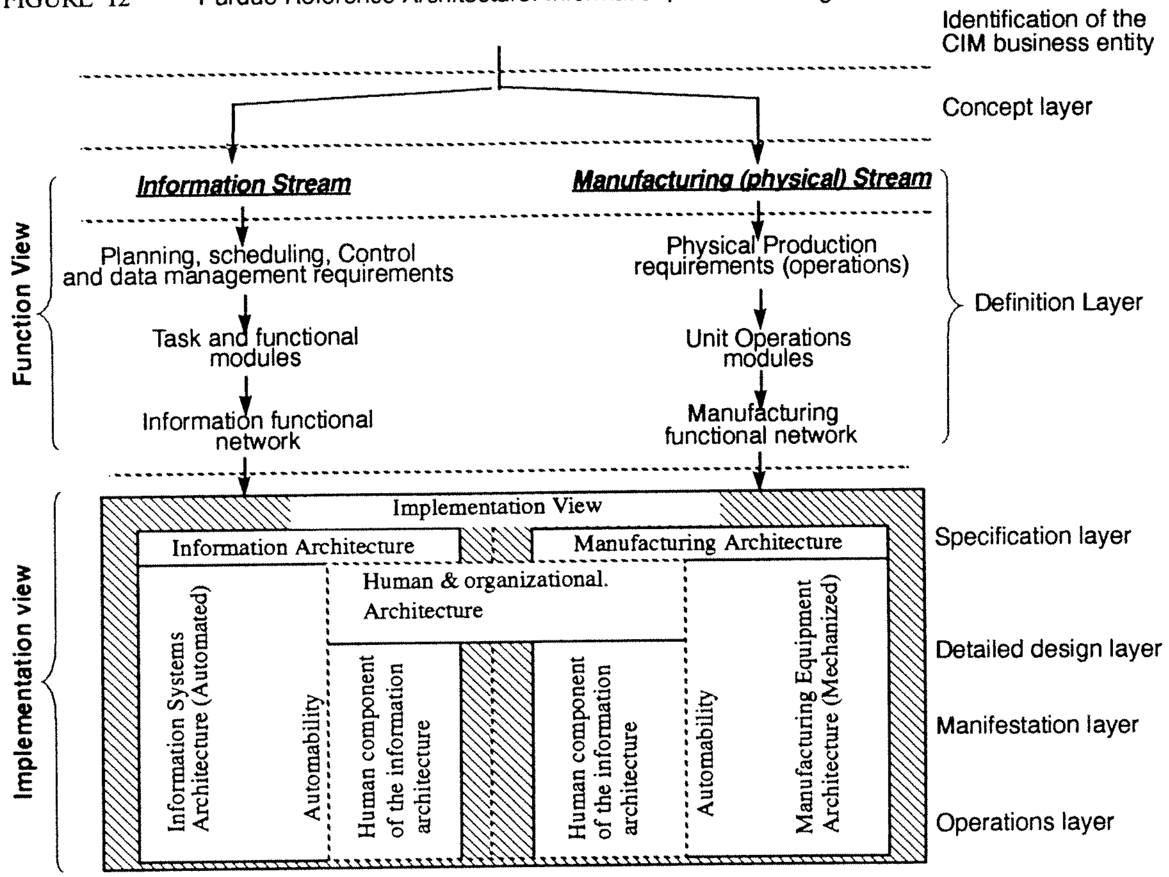


FIGURE 12 Purdue Reference Architecture: Information, Manufacturing and Human Streams


Identification of the CIM business entity
Concept layer
Information Stream
Manufacturing (physical) Stream
Function View
Planning, scheduling, Control and data management requirements
Physical Production requirements (operations)
Definition Layer
Task and functional modules
Unit Operations modules
Information functional network
Manufacturing functional network
Implementation view
Implementation View
Information Architecture
Manufacturing Architecture
Human & organizational. Architecture
Specification layer
Detailed design layer
Manifestation layer
Operations layer
Information Systems Architecture (Automated)
Automability
Human component of the information architecture
Human component of the information architecture
Automability
Manufacturing Equipment Architecture (Mechanized)

The first modelling level, in the function view context, is the setting of both the informational and manufacturing requirements. These requirements tackle issues, from the informational perspective, such as planning, scheduling, control and data management while in the manufacturing stream perspectives operational requirements are set. These functional requirements are transformed into a set of modular tasks which are eventually interconnected to form a network. In the informational tasks, a network represents the data flow between connected function while in the manufacturing context, a network represents a process network.

At the Implementation view: Up to this point, the only modelling streams considered were: informational and manufacturing (physical), but in the implementation view new stream is considered - human/organization architecture (figure 12). Both the informational and manufacturing streams contain tasks that are to be carried out utilizing human skills which, accordingly, needs to be modeled. If the process modelled involves no human skill then only the informational and manufacturing streams are only considered. The degree of human skill involvement in a process is called *Automability* (or *Humanizability*) as shown in (figure 12).

Resources in PERA:

PERA employs a bottom up modelling approach. Accordingly the endeavor has included a generic task and function module. These modules represent a transformation process(es) in a manufacturing environment. These transformations are:

- informational (data) transformation,
- physical (material, energy-based).

Modules can be connected by the flow of material/energy and/or information (data) and/or commands. It is mentioned that the modelling framework of PERA involves modelling of three streams: informational, physical (manufacturing/service) and human/organizational. Accordingly, the definition of *what is* a resource depends on the stream considered.

FIGURE 13 Definition and examples of Task Module

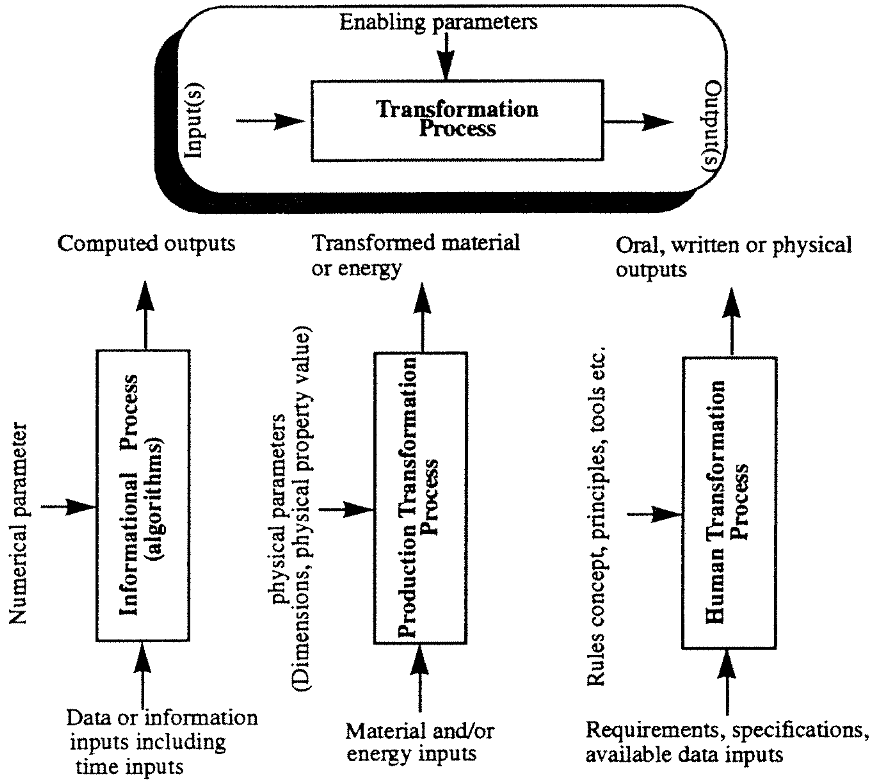

Enabling parameters
Input(s)
Transformation Process
Output(s)
Computed outputs
Transformed material or energy
Oral, written or physical outputs
Numerical parameter
Informational Process (algorithms)
physical parameters (Dimensions, physical property value)
Production Transformation Process
Rules concept, principles, tools etc.
Human Transformation Process
Data or information inputs including time inputs
Material and/or energy inputs
Requirements, specifications, available data inputs

Figure 13 provides the transformation (task) process definition. It includes the definition of the task with its input(s), output(s) and enabling states as modelled in informational, manufacturing and human transformation processes. Information tasks have only informational inputs and outputs. Manufacturing tasks, on the other hand, serve as sources and sinks for information to and from the information tasks. The information needed can be operational, state, commands etc. However in this case data might not be considered as a resource as data is considered as one of the enabling parameters. So resources could be: material, energy, time etc. The human tasks are tasks that require human skills. So the resources considered in the human stream are: employees, requirements, specifications, available data.

2.5 Resource modelling in AI

2.5.1 SIPE [1]

SIPE (System for Interactive Planning and Execution Monitoring) was developed in the eighties with the objective of extending the classical planning[2] paradigm by using causal theories, permitting general constraints and performing replanning. *SIPE* was developed with the intention of having a domain independent methodology for describing a specific domain at different levels of abstraction. A domain includes both performable actions and achievable goals. *SIPE* contains a resource modelling methodology that allows reasoning about resources. It is "viewed as a powerful tool that can be employed by the user to represent domain-specific knowledge concerning the behavior of actions". In SIPE's representation, variables associated with actions are resources.

Resources are classified by being either **reusable** or **consumable**. Reusable resources are used to describe omnipresent resources. The rational behind the implementation of reusable resources is to detect resource conflicts. If an object is defined as reusable resource then no *parallel* action will be allowed to use the resource simultaneously. Parallel interactions are detected by a resource critic and the conflict is resolved through using a system defined heuristic[3].

Consumable resources are used for providing a mechanism for declaring consumption and production of resources. Reasoning about consumable resources is performed through the use of predicates such as: *level, produce* and *consume*. ***Level*** is used for the representation of numerical quantities and is calculated automatically by the system from the predicates describing the production and consumption of quantities. The level predicate occur at certain points (time points) in a plan. It is defined having two arguments: one specifying the object (i.e the resource) and the other specifying the numerical value that is being produced or consumed.

So the two main distinction between reusable and consumable resources in SIPE's context are:

[1] [Wilkins 88]

[2] The classical planning is defined as being state based (i.e taking snapshots of the real world at different time instances.

[3] SIPE detects resource conflicts when in parallel actions an object is declared as a resource. When a conflict is detected by the resource critic it is treated similar to treating harmful parallel action. In general the conflict is resolved through the addition of ordering constraints. For details check [Wilkins 88], chapter 8.

1. The numerical quantity (level) of a consumable resource is either decremented or incremented after the completion of an activity. The level predicate is not used for reusable resources since a reusable resource can only support one activity at a time.
2. A reusable resource can not be used by more than one action simultaneously while it is possible for the case of consumable resources. Accordingly if an object is declared as a reusable resource, with respect to an action, then no parallel action would be allowed to use the same object.

SIPE also specifies that an object could be explicitly defined as being a *shared resource* allowing the object to be assigned to parallel actions.

2.5.2 KRSL

KRSL[1] (Knowledge Representation Sharing Language) is being developed by Knowledge Representation and Architecture Issue Work group in DARP/RL planning and scheduling initiative. The project focuses on the building a sharable and reusable knowledge base. In KRSL's context, a common ontology is "an explicit specification of the ontological commitment of a set of programs" [Fikes et al 92]. The resource ontology has not been fully developed yet. The existing ontology is based on reproduction of previous modelling projects[2].

A resource, according to KRSL, "is an abstract structure defining the capabilities and capacity of various objects and activities".

Resource Types: defines the general properties of a resource such as:

- usage type - is the resource consumed or used?
- is the resource sharable?

There are four types of resources:

1. **consumable resources** such as fuel that will not available after being allocated and the resource capacity is only increased through replenishment of the resource.
2. **reusable/non-sharable** resources that are made available after being allocated. Non-sharable resources are the ones the must be fully allocated to a single activity.

[1] [Allen et al 92].

[2] The project includes modelling of cargo planes.

3. **reusable/synchronized-sharable** are reusable resources that constitute capacity that can support multiple activities simultaneously but "only a temporally synchronized manner". A cargo ship can simultaneously support multiple activities (customers) in condition that the cargo is to be in the ship before a certain time (i.e activities/customers are to use the ship over the "same" interval).
4. **reusable/independently-synchronized** are reusable resource that can be simultaneously allocated to multiple activities without synchronization. Parking is an example of this type where parking activities are performed independent of time.

FIGURE 14 Resource frame in KRSL

```
(define (resource-type <resource-type name>)
      <resource-type description>)

<resource-type description>::=
      [:is-a (<resource-type>*)]
      [:unit-type <unit-type>]
      [:measured-by <list-of-object-attributes>]
      [:allocation-events (<event-rule description>*)]
      [:de-allocation-events (<event-rule description>*)]
      [:production-events (<event-rule description>*))]
```

Figure 14 presents the KRSL resource frame and following is the description of the attached slots.

:unit-type is a default measurement of the resource. It is defined as a "count", "size" or "weight".

:measured-by describes the attributes by which the resource it described. For example a runway may have the attributes *runway-length* and *runway-quality.*

:allocation-events, de-allocation-events, production-events contain a set of events-rules describing how the resource is allocated, de-allocated and produced.

Resource Records defines the provided, required and produced resources by activities. These notions are:

- **provided-resource**: records specify the type and maximum or minimum amount of the resource provided. This record is created relative to a provider object.

- **required-resource**: records are defined relative to a consumer, and the record specifies the type and amount of the resource required/consumed.
- **produced-resource**: records are defined for produced resources specifying the type and maximum or minimum amount of the resource that is producible.

Resource Usage Records contain knowledge about the current resource usage. There are three types of resource records: **provider-usage**, **consumer-usage** and **producer-usage** records. These descriptions are presented in figure 15.

FIGURE 15 Resource records in KRSL - source [Allen et al 92]

Provider usage:
Resource: a record identifying the resource type
Provider: actual provider of the resource
Available capacity: amount of the resource available
Time: time over which that amount is available
Consumer list: consumers of the resource
Producer list: contributors in the production of the resource.

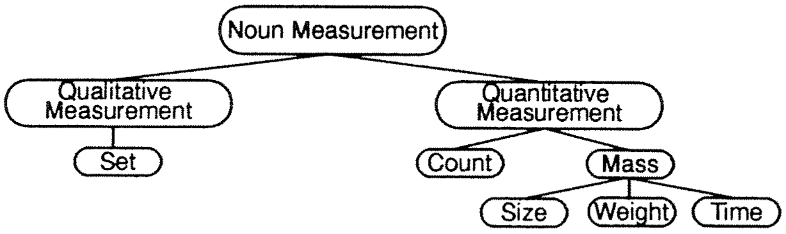
Consumer usage:
Resource: a record identifying the resource type
Consumer: of the resource
Amount consumed: by the consumer
Time: time over which the resource is consumed
Provider list: providers of the resource

Producer usage:
Resource: a record identifying the resource type
Producer: of the resource
Amount produced: by the producer
Time: time over which the resource is produced
Provider list: providers list of producers.

Measurement Primitives in KRSL are:

- **unit-type** defines the unit of measure of quantities. The unit could either in terms of a qualitative set - {large, medium, small} - or in terms of a quantitative type.

FIGURE 16 Unit type in KRSL



- a **quantity** is represented in terms of unit-type and a minimum and/or maximum bound.
- **unit-relation** define the conversions between unit-types.

2.5.3 OPIS Framework for Modelling Manufacturing Systems

OPIS[1] presents a framework for modelling manufacturing systems to be used in knowledge-based simulation and scheduling of manufacturing industries. The model has four basic types of entities: operations, resources, products, demand.

- **Operation entities** are description of activities that are performed in a manufacturing environment.
- **Resource entities** are description of resources that are required to perform manufacturing activities.
- **Product entities** are description of materials that are manufactured by a system.
- **Demand entities** are description of the commitment of a manufacturing industry to deliver products.

OPIS contains a hierarchial description of resources. Moreover, the modeling framework contains a representation of various resource related constraints that affect resource allocation. Figure 17 contains the resource frame as defined in OPIS.

[1] [Smith 90]

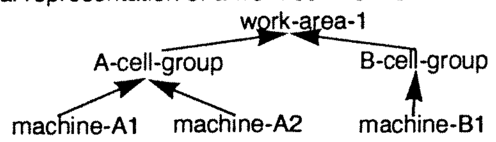
FIGURE 17 Resource frame definition in OPIS

```
{{resource
    IS-A: physical-object
    TYPE:
        range (OR disjoint-aggregation overlapping-aggregation atomic)
    SUB-RESOURCES:
    SUB-RESOURCE-OF:
    OVERLAPPING-SUB-RESOURCE:
    OVERLAPPING-SUB-RESOURCE-OF:
    GROUPED-SUB-RESOURCE:
    GROUPED-IN:
    OVERLAPS-WITH:
    CAPACITY:
    AVAILABLE-CAPACITY:
        range (SET (TYPE instance capacity-interval))
    BREAKDOWN-SPEC:
        range (TYPE instance breakdown-spec)
    REPAIR-LAW:
        range (TYPE is-a repair-operations)
    STATISTICS:
        range (TYPE instance resource-stats-report)
    SCHEDULING-LEVEL:
        range (OR t nil)
    SIMULATION-LEVEL:
        range (t nil)
    DESCRIPTIONS:}}
```

The hierarchial modelling of resources is achieved through using these relations:

- **sub-resources/sub-resource-of**: Are the backbone of the hierarchial resource description. They are used to associate aggregate resources with their constituent resources (and vice versa) under the assumption of mutually exclusive hierarchial decomposition[1].

FIGURE 18 Hierarchial representation of a work-cell - OPIS



- **overlapping-sub-resources/overlapping-sub-resource-of**: These relations are used to express overlapping capabilities of resources. This is used to tackle the case where the allocation of a resource to a "work area" is constraint by the type of product produced. For example, with reference to figure 18, let us assume that a product family **[P]** can only be allocated to "machine-A1" and "machine-B2" in "work-area-1".

[1] Figure 18 presents an example of hierarchial representation in OPIS.

Enjoy your order from <Site Name>!

<URL>

<< Back

Shopping Cart > Shipping > Summary > Billing > Review

Order Confirmation

Thank you for shopping with <Company Name> online!

We've received your order, and will be processing it shortly.

Please take a moment to review your order details. If you have any comments or questions regarding your order, please don't hesitate to contact us at <Company name or customer service>.

Your Receipt

Billing Address

Helen Johnson

3 Wood St.
Toronto
ON M5B 3H6

Credit Card Information

Credit Card Number:	xxxx xxxx xxxx 1111
Credit Card Type:	visa
Credit Card Expiry:	January 2003

Recipient Details

Items being shipped to

John Smith

444 Richmond St.
Toronto
ON M5B 3H6

ORDER NUMBER
yyyy/mm/dd/XXXX

Product Details

Product	[illegible]	[illegible]	Shipping	Subtotal
Product Name Sku Options Quantity: 3	$7.99	$3.99	$3.00	$32.99

It will take 1-2 weeks to deliver to this location using standard shipping.

Helen Johnson

3 Wood St.
Toronto
ON M5B 3H6

ORDER NUMBER
yyyy/mm/dd/XXXX

Product Name Sku Options Quantity: 3	$7.99	$7.99	$3.00	$17.99
Product Name Sku Options Quantity: 3	$7.99	$3.99	$3.00	$32.99

It will take 1-2 weeks to deliver to this location using standard shipping.

Order Subtotal:	$39.89
Shipping:	$14.96
GST:	5.00
PST:	$4.00
Total:	**$56.12**

Once again, thanks for shopping with us! We look forward to serving you again soon.

Accordingly, "machine-A1" and "machine-B2" have overlapping capabilities by being able to support the production of product family **[P]**. "overlapping-sub-resources/overlapping-sub-resource-of" are related to the smallest disjoint aggregation the contains a resource (and vice versa).

- **group-sub-resources/grouped-in**: are also used to define overlapping capabilities. "group-sub-resources/grouped-in" represent an overlapping aggregating which is related to its constituents (and vice versa). That is to say that machines from different cell-groups are grouped because of their similar capabilities.
- **overlaps-with**: "An overlapping aggregation is related to the overlapping aggregations with which it shares constituents via the relation".

Reasoning about resource allocation implies partial allocation of the resource to activities, accordingly OPIS includes capacity-related attributes in the resource modelling framework. These attributes are:

- **capacity**: is defined as to be the number of items that a resource can process simultaneously.
- **batch-size**: specifies the manner in which a resource is allocated[1]. This is used, for example, when a "wash" operation is required to be performed on a "semi-conductors" where they are dipped in a chemical bath for a time interval. The bath could support 50 wafers but if the bath is allocated to a production unit, then no other production unit can use the bath, regardless of the number of wafers already in the bath.
- **current-capacity**: is used in the case when the allocation of resources is done in strictly time ordered manner (i.e in the process of a forward simulation) with no future anticipations.
- **capacity-interval**: is a time interval in which available capacity of a resource is described.
- **available-capacity**: is used when future demand and plans are incorporated, then the representation depicts the evolution of each resource's available capacity over a future time horizon. The available capacity is expressed in terms of ordered sequence of capacity-intervals

Requirements of manufacturing activities (operations) in OPIS are defined through **resource-requirement-spec**. The specification is defined with two attributes:

[1] i.e a constraint on the allocation of the resource to an activity

- **required-resource-fun** which specifies the resource sub-pool (an aggregate resource).
- **required-quantity-fun** which defines the amount of the resource's capacity required.

FIGURE 19 resource requirement specification definition - OPIS

```
{{resource-requirement-spec
   IS-A: conceptual-object
   REQUIRED-RESOURCE-FUN:
   REQUIREED-QUANTITY-FUN:}}
```

Resource setup constraints are also modelled in OPIS through the use of a *setup matrix* which specifies the setup duration constraint. The setup time related attributes are:

- **setup-matrix**: specifies the setup duration constraints.
- **config-dependent-setup-matrix**: specifies the setup durations in terms of the time needed to change the resource's configuration from the that imposed by the previous operation to the configuration imposed by the current operation.
- **location-dependent-setup-matrix**: is similar to config-dependent-setup-matrix with the addition of the time to move the resource form one location to another.

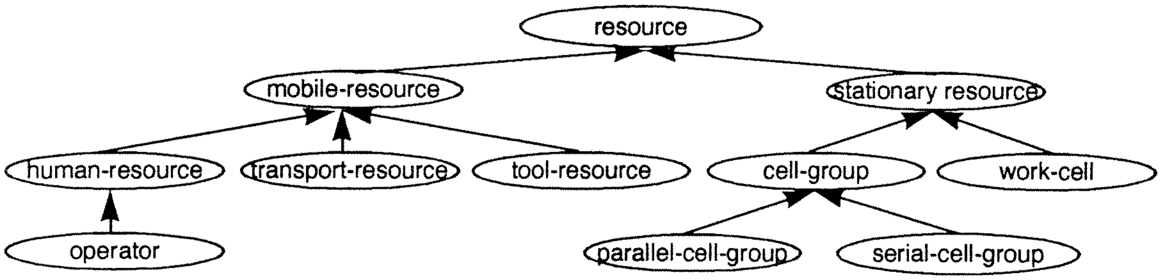
Modelling of work shift's specification is done through defining time periods through which the resources are operational. This is defined using **work-shift-spec** which in essence provide a solution of representing a time varying constraints. The work shift constraint is defined over a delineation of time intervals over a specific time horizon[1].

Another attribute of a resource defined specifies the **breakdown characteristics.** These characteristics are: time to next failure and amount of capacity lost. These characteristics are represented through using breakdown-spec. A related attribute is the **repair-law** attribute the specifying the repair specification using **repair-operations.**

[1] A work shift specification is defined as calender intervals containing description of work shifts. A **shift** is defined as an **hours-of-day-interval,** associated with a work-week which in turn is defined as **days-of-week-interval.**

In OPIS, resources are delineated to being either **stationary** or **mobile** ones. The taxonomy is presented in figure 20.

FIGURE 20 Taxonomy of resources in OPIS based mobility characteristics



Finally, the last attributes in the resource frame - figure 17 - that have not been described yet are:

- **statistics** which a repository for resource utilization statistics,
- **scheduling-level** and **simulation-level** are 'markers' to indicate the level of precision.

A related entity to resource modelling is the product entity. A *product* refers to an object internally manufactured by a system, either as finished or semi-finished objects. Similar to resource modelling framework, products are hierarchial model. This is done through defining a *product family* to represent a collection of products whose manufacturing process is the same. Defining the assembly/subassembly and aggregate/disaggregate relationships, of a product, is achieved through the use of **material-requirements/material-requirement-for** relationships. In the of *material-requirement-for* relationship a meta-information is attached to it, quantity-spec, specifying the amount of material that is required for the production of an object. It also specifies the number of sub-components needed for the production of a composite product.

2.5.4 Gerry

Gerry[1] is a scheduling and rescheduling system that uses constraint based iterative repair to produce schedules that does not violate resource and state constraints. Gerry is used by NASA in managing Space Shuttle processing. Gerry contains a constraint modelling language. These constraints are: temporal/milestone, resource and state constraints.

[1] [Zweben et al 93]

Gerry models resource constraints in terms of classes and pools. "A *class* represents a type of resource consisting of a set of resource pools. A *pool* represents a collection of indistinguishable resources". Figure 21 shows an example, from the Space Shuttle domain, of the use of classes and pools: *heavy-equipment* class contains the pools *crane-crew* and *high-crew*. Each pool is initially defined with maximum amount of the resource that is available. The availability of the resource is maintained through maintaining the *history* for each pool.

FIGURE 21 Example of a class and a pool - Gerry

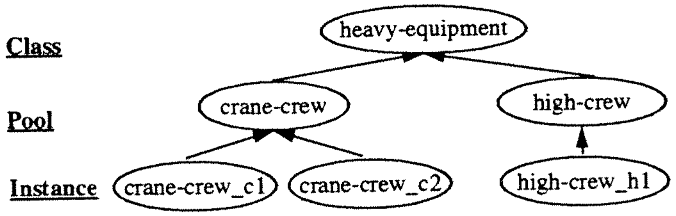
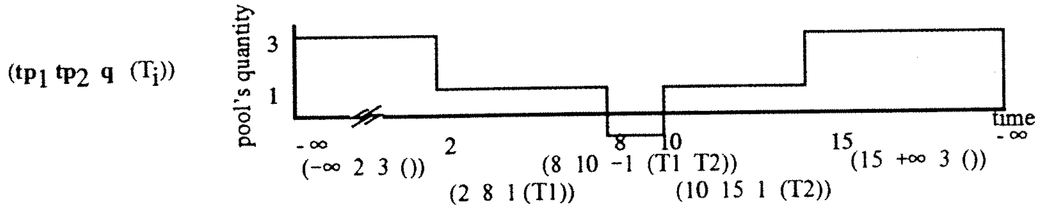


FIGURE 22 Resource history - Gerry



The history of resource pools is defined as a tuple; the arguments of the tuple are: the starting time of the tuple (tp1), the end time of the tuple (tp2), the quantity of the pool available in the specified time period (q) and the list if activities supported by the resource ($T_{i\circ n}$)

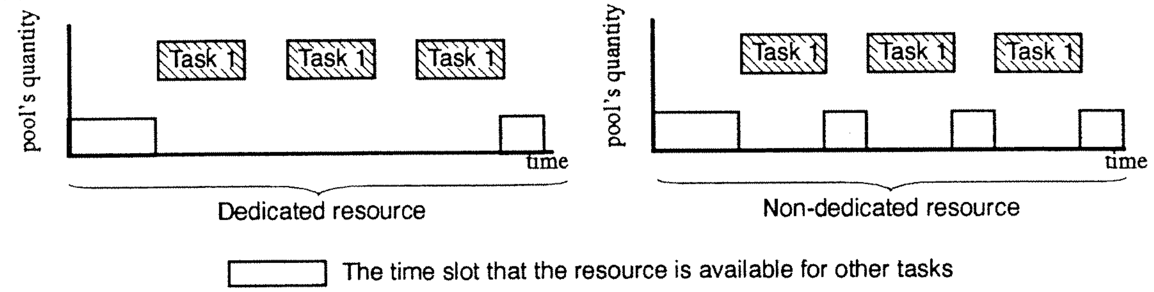
Figure 22 presents an example of the application of resource history. For example, the second history tuple specifies that the resource for time two to eight had only one unit and that the resource supported one task - T1. Through the use of the activity history, capacity violation is detected similar to the presented example where the capacity constraint is violated from eight to ten time points. Moreover the "resource pool quantity" is incremented and decremented in discrete manner while linear change is not supported.

Tasks requirements specify a minimum set of resources associated with amount needed. Each resource requirement consists of type and quantity and is represented as a tuple that spans a task. As an example:

- (T_1 2 R_1 Tp_1 Tp_2) specifies that task T_1 requires two of R_1 resource from that start-time (Tp_1) to the end-time (Tp_2) of the task.

A resource could also be described as being *dedicated* or *non-dedicated.* A resource is said to be dedicated to a task, when the resource can not be allocated to a second task even if the resource has the capacity.

FIGURE 23 Dedicated vs. Non-dedicated resources - Gerry



Resource's quantity is decremented/incremented in discrete manner, over the time period of the task, in case the resource is consumed. Gerry presents two options of decrementing/incrementing the resource's quantity:

- starting from the start-time of the task to infinity,
- starting from the end-time of the task

State constraints model attributes that change over time and might have several possible values. Similar to what is discussed in resource modelling, each task specifies ***state requirements*** designating the state required. For instance, a task may require that the value of attribute of *Power-button* be *ON* for its start to end times. The state requirement could also be the position of *left-payload-bay-door* of the shuttle or the configuration of a certain tool.

2.5.5 CYC

CYC[1] is a project whose aim is to build a knowledge base to support natural language understanding and machine learning. A goal of the project is to provide Cyc with enough common-sense knowledge to enable it to read and understand an encyclopedia article. The ontology defined in Cyc is basically organized around categories of things in the world. The categories are arranged in a generalization/ specialization delineation having "thing" as the root- figure 24 & 25.

[1] [Guha et al 90]

FIGURE 24 Taxonomy of Ontology - Cyc

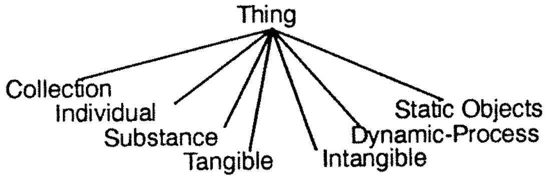
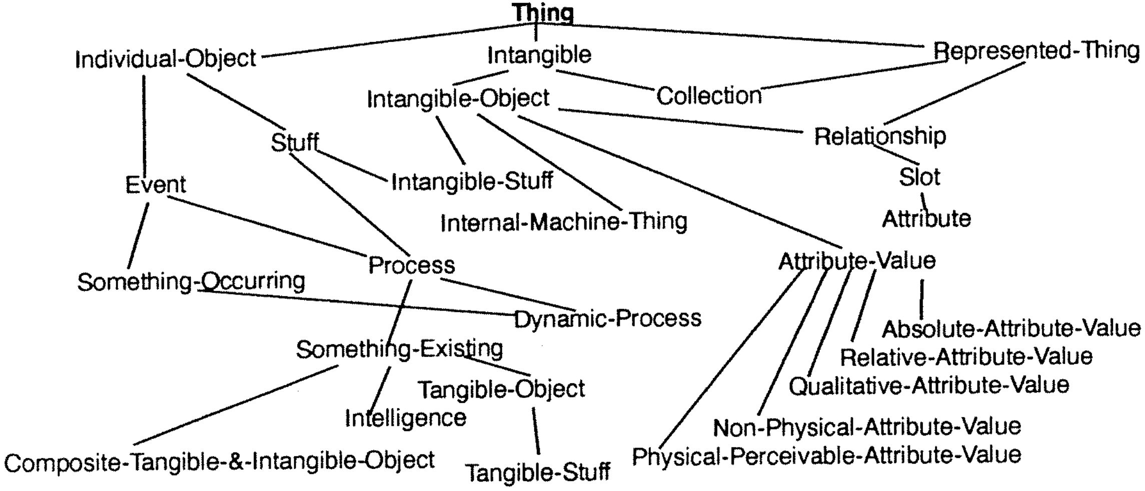


FIGURE 25 Taxonomy of Ontology cont'd - Cyc



"Thing" is used for modelling the world by being partitioned in three ways:

- Represented-Thing vs. Internal-Machine-Thing
- Individual-Object vs. Collection
- Intangible vs. Tangible-Object vs. Composite-Tangible-&-Intangible-Object.

Substance (stuff) vs. Individual-Object: Substances are things that when divided, will result in a number of things similar to the original object. Wood and water are examples of a substance. Instances of wood and water would inherit their *intrinsic properties*. Individual objects on the other hand are objects when divided will not result multiple copies of the objects. A monitor and an axe are examples of individual objects. "Substance" and "Individual-Objects" are in turn instances of "Substance-Type" and "Object-Type" restrictively.

Events vs. Processes: An event is a set of things with temporal relationships (before, after etc.). A process is an event with the difference that a temporal slice of the process is still a process while a temporal slice of an event is not the same event anymore. Playing two games of tennis is not a process but it is an event while playing tennis is a process. Similar to the approach in "substance" and "individual-object", an "event" and a "process" are instances of "event-type" and "process-type" respectively.

Quantity: The approach used to describe quantities is interval-based. Example of such intervals are "around 180 pounds". The specified interval could be an open-ended one -"more than 180 pounds". To deal with the accuracy of representation of the interval, another interval is specified in which it is believed the actual length of a unit exist (10 plus or minus 0.5).

Agents: An agent is a sub-set of "Composite-Tangible-&-Intangible-Object" and it represents units that are capable of taking decision making actions, controlling and changing situations. Agents include people, computer programs etc. Moreover agents can be collective such as organization and institutions. In order for the agents to be performed they need to "own" resources.

Resource: "A resource is anything that can be a tender in some transaction." In other words the term resource represent that an object is owned by an agent. An example of a context in which a resource is represented in is: "In transaction T1, there are agents A1 and A2 and there is some resource R such that there is a change in the relationship Q between R and A2 after T1.

$$\text{resource-transfered}(T1, R) \wedge \text{instance-of}(T1, C) \wedge \text{from-agen}(T1, A1) \wedge \text{to-agent}(T1, A2)$$

$$\wedge\ \text{transfer-type}\ (T, Q) \rightarrow Q(A2, R))\text{"}$$

So agents require resources. These resources could be information, time, effort, space, energy (etc.) {Blair et al 92].

Structure: Any tangible object is structured if it could be broken down to sub-parts. In much the same way, an action is structured if it has sub-events.

2.6 Summary

Among the enterprise reference models considered - IWI, CIM- OSA, PERA - IWI is the most developed from a resource modelling perspective. Equipment, tools, personnel and parts/structures are considered as resources in the IWI model. The IWI model has defined attributes describing different resource perspectives such as cost, technical-physical, classification, state, planning and maintenance data. Moreover, the IWI model has defined attributes to represent a parts/structures, equipment group, equipment, tool and personnel. Capacity is represented as an attribute in both the "equipment" and "tool" entities. Equipment's capacity is defined in terms of periods (time) of capacity before being phased out. This implies that an equipment in IWI has the capacity of supporting one job at a time. Capacity is defined in terms of hours of usage of the resource. Similarly, tool's capacity is modelled in terms of hours of assembly. As for the personnel entity, no attribute is defined that portrays the respective employee's capacity.

A resource in CIM-OSA's context is a reusable one while a consumable resource is defined as a function input. CIM-OSA has the intention of defining a "resource view" however that has not been done yet. Still CIM-OSA has defined several constructs in the different modelling levels that specify the alternative, specified and implemented resources in an application. More over, CIM-OSA has defined two grouping constructs, logical and organizational cell, that serve as a resource pool. Resource's capabilities are modelled through the related resource constructs defined over the modelling levels. These constructs are *specified* and *implemented capabilities*. However, it is not mentioned the means of describing these capabilities.

As for PERA, its resource model is embedded in the framework but lacks a data model. PERA's task and function modules are similar to that defined in IDEF$_0$ formalism where each task/function requires inputs (i.e requires resources) and some enabling parameters to be performed. It has been shown that the inputs and the enabling parameters depend on the context in which it is considered (i.e informational, manufacturing or organizational/human).

Among the endeavors surveyed from the AI domain, OPIS is the most developed one. Also OPIS has defined resource aggregation relations that are used to define overlapping capabilities. Moreover, OPIS contains breakdown, set-up time and resource requirement specifications. A resource could either be mobile or stationary. Capacity is modelled in OPIS through the "capacity" and "available capacity" attributes in the resource frame. Capacity is modelled as a function of the resource's capacity constraints. Some resources are defined having the ability to support multiple jobs[1] simultaneously.

Gerry and Sipe were built to perform scheduling and planning activities. Both endeavors have defined a resource as being either a reusable or consumable. Reusable resources in Gerry can not support multiple jobs. The same is true in SIPE were a reusable resource can not support parallel actions.s As a matter of fact, resource conflicts are detected if a resource is assigned to parallel actions[2]. Gerry

[1] i.e "production units" as defined in OPIS.

has further defined the notion of "resource history" for a reusable resource specifying the resource quantity and the activity's that are supported over a specified time interval. Finally a resource in Gerry is defined as wither being dedicated or non-dedicated with respect to a task.

In KRSL, a resource "is an abstract structure defining the capabilities and capacity of various objects and activities". A resource is defined as being consumable, reusable/non-sharable, reusable/synchronized-sharable or reusable/independently-synchronized. A resource frame is defined in KRSL which specifies the type of resource, unit of measurement, rules of allocation and de-allocation. KRSL has the notion of having a reusable and sharable resource however no modelling framework is presented.

Finally Cyc's domain is general knowledge understanding and not manufacturing one and hence did not define the required ontology needed to perform manufacturing activities. Still Cyc has defined that in order for "agents" to performed they have to own "resources". "A resource is anything that can be a tender in some transaction". However no explicit definition of resources and their attributes is defined.

2.7 Conclusion

What is needed is to incorporate a generic resource ontology (model) in the context of manufacturing enterprise so that the ontology could be used by different manufacturing, design, accounting, sales (etc.) activities. This is to be done through merging concepts (terms), already defined in the reviewed research, with some new ones to reach to a generic resource model.

Terms should have a well formalized and rigorous definition (in the form of axioms) to limit ambiguities. The model should also support the ability to deductively capability to answer simple questions so that to turn the resource reasoning into an open deductive process as opposed to specialized code embedded in an application, thereby making it shareable/reusable. That is, for the generic model to be sharable, not only ontology should be provided but also the reasoning processes associated with it.

[2] unless the resource is **explicitly** defined as a "shared resource"

CHAPTER 3 *Ontologies for Enterprise Integration [TOVE]*

This chapter presents a logical framework for representing activities/processes, states, and time in an enterprise integration architecture. This framework provides the basis for integrated supply chain management and enterprise engineering in the Toronto Virtual Enterprise project at the University of Toronto.[1]

[1] This chapter is a reprint from [TOVE 94].

3.1 Introduction

Enterprise modelling is an essential component in defining the tasks and functionality of the various components of an enterprise.The goal of our enterprise modelling research is to create a generic, reusable representations of Enterprise Knowledge that can be reused across a variety of enterprises. Towards this end, we have been developing the TOVE (Toronto Virtual Enterprise) ontology [Fox et al 93a] and applying this ontology to enterprise engineering [Fox et al 94], enterprise integration, and integrated supply chain management. An ontology is a formal description of entities and their properties; it forms a shared terminology for the objects of interest in the domain, along with definitions for the meaning of each of the terms. TOVE provides a rich and precise representation of generic knowledge, such as, activities, processes, resources, time, and causality, and of more enterprise oriented knowledge such as cost, quality and organization structure.

The basic entities in our model are represented as objects with specific properties and relations. Objects are structured into taxonomies. Definitions of objects, attributes and relations are specified in first-order logic, where possible. We then define an ontology in the following way. We first identify the objects in our domain of discourse; these will be represented by constants and variables in our language. We then identify the properties of these objects and the relations that exist over these objects; these will be represented by predicates in our language.

We next define a set of axioms in first-order logic to represent the constraints over the objects and predicates in the ontology. This set of axioms constitutes a micro-theory ([Guha et al 90]) and provides a declarative specification for the various tasks we wish to model.

Intuitively, the axioms in the microtheory enable the model to deduce answers to questions that one would normally assume can be answered if one has a "common-sense" understanding of the enterprise. To formalize this intuition we also need to prove results about the properties of our micro-theories in order to provide a characterization and justification for our approach; this enables us to understand the scope and limitations of the approach. We use a set of problems, which we call competency questions, that serve to characterize the various ontologies and micro-theories in our enterprise model. The micro-theories must contain a necessary and sufficient set of axioms to represent and solve these questions. It is in this sense that we can claim to have an adequate microtheory appropriate for a given task, and it is this rigour that is lacking in previous approaches to enterprise engineering and integrated supply chain management.

3.1.1 Requirements for an Ontology of Activity and Time

Each of the following requirements provides a set of competency questions.

- We need to evaluate the truth value of a proposition at some point in time in the future. We therefore need to define axioms that express how the truth of a proposition changes over time. In particular, we need to express the properties and relations that change or do not change as the result of an activity.
- We must define the notion of a state of the world, that is, define what is true of the world before and after performing different activities. This is necessary to express the causal relationship between the preconditions and effects of an activity.
- The interval over which the state has a certain status is bounded by the times at which the appropriate actions that change status occur. This interval defines the duration of a state if the status is enabled.
- We want a uniform hierarchical representation for activities (aggregation). Plans and processes are constructed by combining activities. We must precisely define how activities are combined to form new ones. The representation of these combined activities should be the same as the representation of the sub-activities. Thus aggregate activities (sets of activities or processes) should themselves be represented as activities.

- The causal and temporal structure of states and sub-activities of an activity should be explicit in the representation of the activity.

3.2 Time

In this section we define the ontology of time that is used in this work. This includes defining the objects in the domain and relations over these objects, as well as defining the relationship between time and actions, which will be used throughout the remainder of the paper.

3.2.1 Time Points and Intervals

We represent time as a continuous line; on this line we define time points and time periods (intervals) as the domain of discourse.

We define a relation < over time points with the intended interpretation that $t < t'$ iff t is earlier than t'. We represent uncertainty by defining the following predicate:

$time\_point(t, min, max) \equiv min \leq t \leq max$

The functions $SP(t)$ and $EP(t)$ denote the start and end points of the interval t, respectively. Using the relation < and these functions, we can define relations over intervals based on the temporal ordering of the endpoints of these intervals extending the relations of [Allen 83]. For example, we can define the relations

$$(\forall\ t,t',p_1,p_2,p_1',p_2')\ strictly\_before(t,t') \equiv$$

$$time\_point(EP(t), p_1,p_2) \wedge time\_point(SP(t'), p_1',p_2') \supset p_2 < p_1'$$

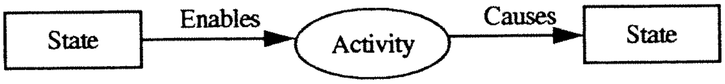
$$(\forall\ t,t',p_1,p_2,p_1',p_2')\ possibly\_before(t,t') \equiv$$

$$time\_point(EP(t), p_1,p_2) \wedge time\_point(SP(t'), p_1',p_2') \supset p_2 > p_1' \wedge pl < p_2'$$

3.3 Activities and States

At the heart of the TOVE Enterprise Model lies the representation of an *activity* and its corresponding enabling and caused *states*. An activity is the basic transformational action primitive with which processes and operations can be represented; it specifies how the world is changed. An enabling state defines what has to be true of the world in order for the activity to be performed. A caused state defines what is true of the world once the activity has been completed. We will begin by presenting states, and then define how properties of activities are defined in terms of these states.

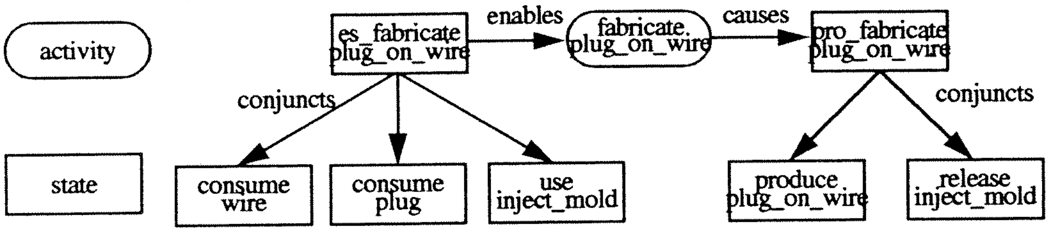
FIGURE 26 Activity-State Model



3.3.1 States

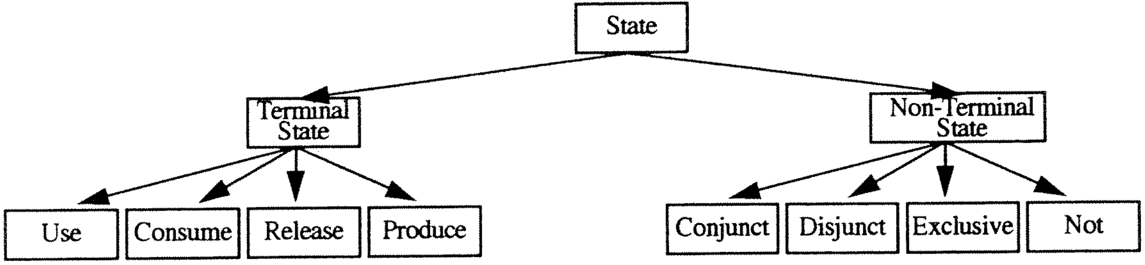
An activity, along with its enabling and caused states, is called an *activity cluster.* The state tree linked by an *enables* relation to an activity specifies what has to be true in order for the activity to be performed. The state tree linked to an activity by a *causes* relation defines what is true of the world once the activity has been completed. Intermediate states of an activity can be defined by elaborating the activity into an activity network as we will define later in the paper.

FIGURE 27 Activity-State Cluster



There are two types of states: *terminal* and *non-terminal.* In Figure 2, **es_fabricate_plug_on_wire** is the non-terminal enabling state for the activity **fabricate_plug_on_wire** and **pro_fabricate_plug_on_wire** is the caused state for the activity. The terminal substates of **es_fabricate_plug_on_wire** are **consume_wire, consume_plug,** and **use_inject_mold**; the terminal states of **pro_fabricate_plug_on_wire** are **produce_plug_on_wire** and **release_inject_mold.**

FIGURE 28 State Taxonomy



At present we recognize four terminal states represented by the following predicates:

use(s,a), consume(s,a), release(s,a), produce(s,a)

These predicates relate the state with the resource required by the activity. Intuitively, a resource is used and released by an activity if none of the properties of a resource are changed when the activity is successfully terminated and the resource is released. A resource is consumed or produced if some property of the resource is changed after termination of the activity; this includes the existence and quantity of the resource, or some arbitrary property such as color. Thus *consume(s,a)* signifies that a resource is to be used up by the activity and will not exist once the activity is completed, and *produce(s,a)* signifies that a resource, that did not exist prior to the performance of the activity, has been created by the activity. We define use and consume states to be enabling states since the preconditions for activities refer to the properties of these states, while we define release and produce states to be caused states, since their properties are the result of the activity. We will later show how the state predicates can be formally defined in terms of the effect axioms for actions in the theory.

Terminal states are also used to represent the amount of a resource that is required for a state to be enabled. For this purpose, we introduce the predicate *quantity(s,r,q)*, where *s* is a state, *r* is the associated resource, and *q* is the amount of resource r that is required. Thus if *s* is a consume state, then *q* is the amount of resource consumed by the activity; if *s* is a produce state, then *q* is the amount of resource produced.

We assign values to states to capture the notion of status. We define a new sort for the domain of the status of a state with the following set of constants:

[possible, committed, enabled, completed, disenabled, reenabled]

Non-terminal states enable the boolean combination of states. We will consider four non-terminal states:

conjunctive, disjunctive, exclusive, not

Disjunctive states are used to formalize the intuition of a resource pool. We may have a set of resources, such as machines or operators, that can possibly be used by an activity. The activity only requires one of these resources, so the activity only needs to non-deterministically choose one of the alternative resources in the pool. Thus, the status of the disjunctive state changes if one of the resources has been selected and its status has been changed.

The occurrence of an action for a conjunctive state is equivalent to the occurrence of the action for all of its substates. The occurrence of an action for an exclusive state is equivalent to the occurrence of the action for exactly one of the substates. The occurrence of an action for a not state is equivalent to the non-occurrence of the action for its substate.

3.3.2 Activities

An activity specifies a transformation on the world. There are several primitive pieces of knowledge associated with it. The following enumerates the attributes of an activity:

We define a new sort for the domain of the status of an activity with the following set of constants:

[dormant, executing, suspended, reExecuting, terminated]

The status of an activity is defined by the status of its enabling and caused states. The complete logical definitions for the status of activities and states can be found in [TOVE 94]. In this thesis, the predicate *activity(a, status, tp)* is used to denote the status of activity *a* at time point *tp*.

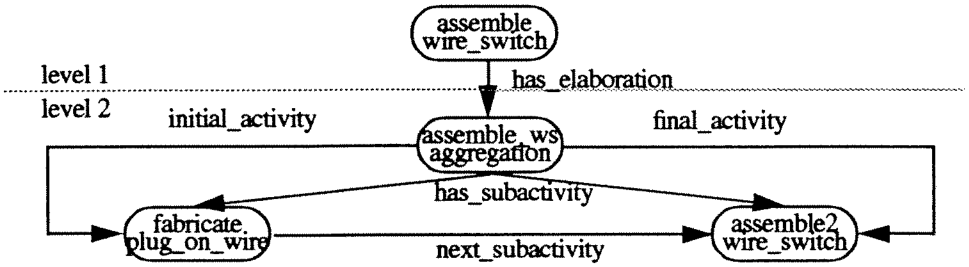
3.3.3 Duration

By combining the ontology of time with the ontology of states of activities, we arrive at the notion of duration. The duration of a state is defined as the time period beginning at the time that the state is enabled and ending at the time that the state is completed., Similarly, the duration of an activity is defined as the time period beginning at the time that activity begins the status of executing and ending at the time that the activity begins the status of terminated. The duration of a state is represented by the predicate *state_duration(s, d)*, while the duration of an activity is represented by the predicate *activity_duration(a, d)*.

3.4 Aggregation and Abstraction of Activities

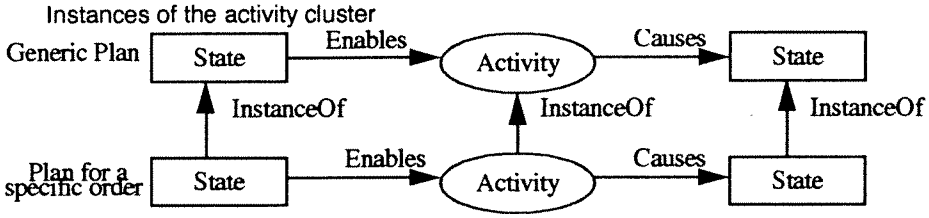
An important requirement for an ontology for activities is the ability to aggregate a set of activities to form a new activity. Activity clusters may be also aggregated to form multiple levels of abstraction. "An activity is elaborated to an aggregate activity (an activity network), which then has activities" [Sathi et al 85]. These activities are sub-activities of the aggregate activity (see FIGURE 29).

FIGURE 29 Activity Abstraction



The enabling and caused states are omitted from this diagram but they do exist. Just as activities can be abstracted, states can be abstracted in a similar manner. We can similarly define classes of activities and instances of activities in these classes:

FIGURE 30 Instances of the activity cluster



We introduce the predicate *subactivity(a, a')* to denote that activity *a'* is a sub-activity of activity *a*. In the same way that we defined the temporal relations over the substates of a non-terminal state, we define temporal relations over the sub-activities of an aggregate activity. The formalization of these relations remains for future work.

3.5 TOVE Factory

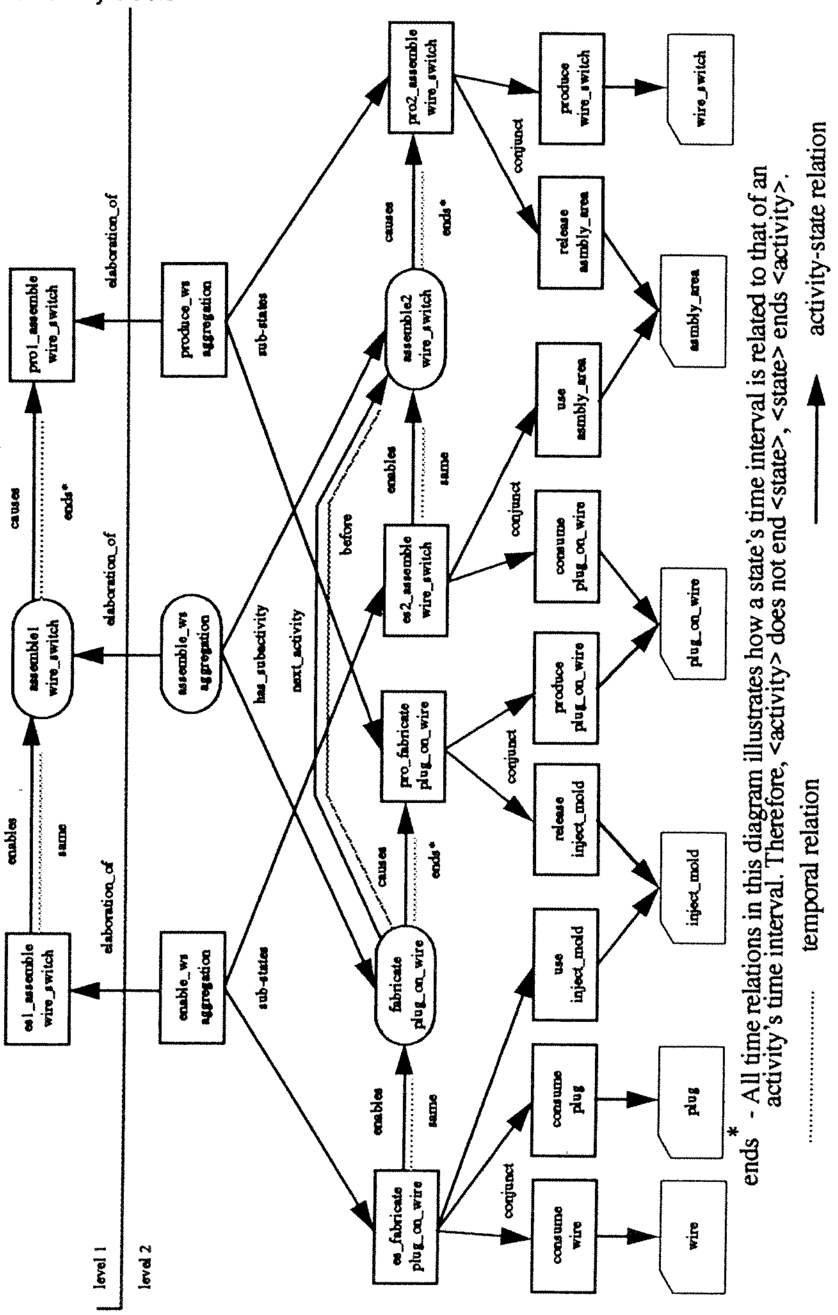
A "virtual factory", called TOVE, has been defined using the ontologies in this paper. We now present an example to illustrate the modelling of two activities and their abstraction. The two activities are fabricate plug_on_wire and assemble2 wire_switch. We have already introduced the activity cluster for fabricate plug_on_wire (see Figure 27) and the abstraction of fabricate plug_on_wire and assemble2 wire_switch without the corresponding states (see Figure 29). The activity clusters, their abstraction, and corresponding states are now illustrated in Figure 31.

fabricate plug_on_wire is enabled by the consumption of plug (**consume plug**) and the use of inject_mold (**use inject_mold**). This activity causes the production of plug_on_wire (**produce plug_on_wire**) and the release of the used resource

(release inject_mold). The enabling state of fabricate plug_on_wire (**es_fabricate plug_on_wire**) is a subclass of a conjunct state since all three resources must be present for the activity to occur. Similarly, the cause state of fabricate plug_on_wire (**pro_fabricate plug_on_wire**) is a subclass of a conjunct state.

assemble2 wire_switch is enabled by the consumption of plug_on_wire (**consume plug_on_wire**) and the use of an assembly area (**use asmbly_area**). This activity causes the production of wire_switch (**produce wire_switch**) and the release of the used resource (**release asmbly_area**). The enabling and caused states of assemble2 wire_switch are also subclasses of the conjunct state.

FIGURE 31 plug_on_wire activity cluster



* ends - All time relations in this diagram illustrates how a state's time interval is related to that of an activity's time interval. Therefore, <activity> does not end <state>, <state> ends <activity>.

temporal relation

activity-state relation

fabricate plug_on_wire and **assemble2 wire_switch** are sub-activities of **assemble_ws aggregation. es_fabricate plug_on_wire** and **es2_assemble wire_switch** are substates of **enable_ws aggregation. pro_fabricate plug_on_wire** and **pro_assemble wire_switch** are substates of **pro_ws aggregation**.

3.6 Conclusion

This chapter presented a logical framework for representing activities/processes, states, and time in an enterprise integration architecture. This framework provides the basis for integrated supply chain management and enterprise engineering in the Toronto Virtual Enterprise project at the University of Toronto.

Reasoning about the activities/states and their status can not be achieved unless a resource reasoning framework is available. The complexity of planning and scheduling is determined by the degree to which activities contend for resources. Allocation of resources, defining the resource's capacity are examples of such reasoning processes. What is presented in the coming chapter is the formalization of ontology required to model enterprise resources that enables a system to reason about the status of activity and its resources.

CHAPTER 4 *Resource Ontology*

The purpose of this chapter is to define the required resource for the use in planning and scheduling. The ontology is defined using first order logic and then implemented in Prolog to give the ontology the capability of deductively answering queries.

4.1 Introduction:

In this chapter an ontology for modeling and representing resources is presented. The ontology is generic and applicable to many domains. Furthermore most terms are defined in first order logic (FOL). FOL is used because of its expressive and declarative capability. With the ontology defined in FOL and implemented in a theorem prover, the ontology would have the deductive capability to answer queries. Accordingly, the ontology provides a general descriptive language with which reasoning about the world is performed.

The competency of the ontology is defined by a set of questions[1] that the representation should be able to answer. The *competency* of an ontology represents the extent to which it supports problem solving. These questions represent the basic accesses a problem solver would make to the representation. However the competency questions do not only represent simple retrievals from a knowledge base, but also entail deduction. The deduction operation uses Prolog version of the FOL definition of each ontological term. Recall from chapter one, examples of competency questions are:

- **Quantity:** How much of the resource exists at time t?

[1] defined in chapter six.

- **Consumption:** Is the resource consumed by the activity? If so, how much?
- **Divisibility:** Can the resource be divided and still be usable? Can two or more activities use the resource at the same time?
- **Structure:** What are the subparts of resource R?
- **Capacity:** Can the resource be shared with other activities?
- **Location:** Where is resource R?
- **Commitment:** What activities is the resource committed to at time *t*?
- **Trend:** What is the capacity trend of a resource based on the machine usage history?

In chapter two, various endeavors from the reference models/manufacturing domain and AI domain are presented. Most of the requirement found in these models are integrated in the ontology.

Resources are:

Generally speaking, a resource is an object which has to be available at the time of action of an activity. In TOVE, a *resource* denotes a role of an entity/object in an activity rather than just being as subclass of an entity - resource. Therefore an object is not considered as a resource if it is not associated with an activity. The properties of a resource are determined by the role in an activity.

Accordingly, resources could be:

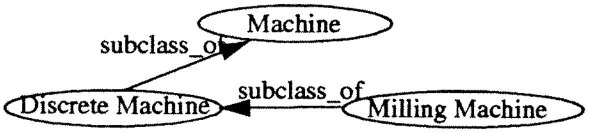
- **machines** such as milling machines, when associated when associated with milling activities
- **materials** such as raw material, semi finished products consumed in an assembly or manufacturing activity
- **tools** such fixtures, cranes, chairs used by an activity
- **human skill** that is needed to perform an activity,
- **floor space** that is used by an activity,
- **electricity** when consumed by an activity,
- **capital** when consumed of an activity,
- **information** communicated between enterprise units.

The resource ontology defined in this chapter, is built on TOVE's existing activity-state and time ontology[1]. In turn the activity-state and time ontology are based on object oriented representation with objects organized into taxonomies. "subclass of" and "instance of" are primitive relations. "subclass of" defines a subclass/ superclass relationship. A subclass is a specialization of the superclass and hence the subclass inherits the superclass attributes. The subclass_of predicate is defined as a binary ground term with the following parameters:

- **R**: Resource name.
- **Y**: The parent class.

```
subclass_of(R, Y).
```
(PRO 1)

FIGURE 32 subclass_of



```
subclass_of(milling_machine, discrete_machine).
```
(PRO 2)

A *milling_machine* is subclass of a *discrete machine.*

```
subclass_of(vehicle, transportation_resource).
```
(PRO 3)

Resource *vehicle* is subclass of a *transportation_resource.*

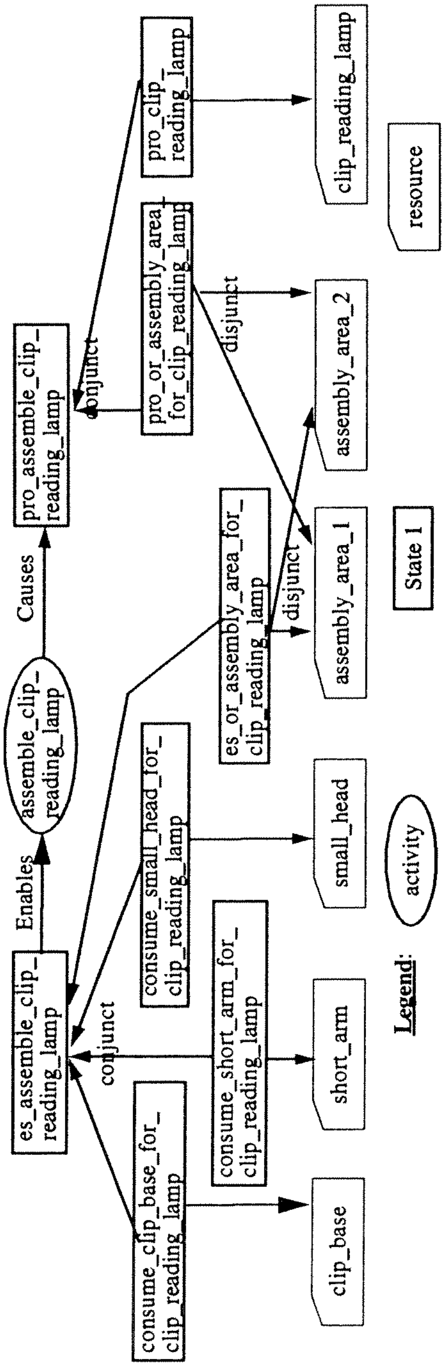
"instance of" relation (instance_of(x, y)) specifies a real object.

As previously mentioned, the resource ontology is based on the *activity/state ontology*, which is the heart of the TOVE representation, and is composed of *activity* and its corresponding *enabling and caused states*. Throughout this chapter, ontology of activity-state and time will be used[1].

[1] as defined in chapter three

[1] See appendix E for time points and periods relations and data base.

FIGURE 33 Assemble_clip_reading_lamp cluster [TOVE 1992]



es_assemble_clip_reading_lamp
Enables
assemble_clip_reading_lamp
Causes
pro_assemble_clip_reading_lamp
conjunct
pro_or_assembly_area_for_clip_reading_lamp
pro_clip_reading_lamp
disjunct
clip_reading_lamp
assembly_area_2
assembly_area_1
es_or_assembly_area_for_clip_reading_lamp
disjunct
consume_small_head_for_clip_reading_lamp
small_head
consume_short_arm_for_clip_reading_lamp
short_arm
conjunct
consume_clip_base_for_clip_reading_lamp
clip_base
Legend:
activity
State 1
resource

The activity 'assemble_clip_reading_lamp', figure 33, will be used as an example throughout this chapter. *State 1* represents the requirement(s) of the activity which sets what has to be true for the activity to be performed while *state 2* specifies that *assembly_area_1* will be released and *clip_reading_lamp* will be produced after the completion if assemble_clip_reading_lamp activity.

Legend:

- **R:** Identification name [ID] of resource R.
- **A:** ID of activity A.
- **U, Unit**: unit of measurement
- time_point(Time_point_ID, Min, Max)

  A *time point* (tp) is the most basic time representation. It consists of identification name of the time point, minimum and maximin time.
- time_period(Time_interval_ID, ST, ET, MinDur, Dur, Max Dur)

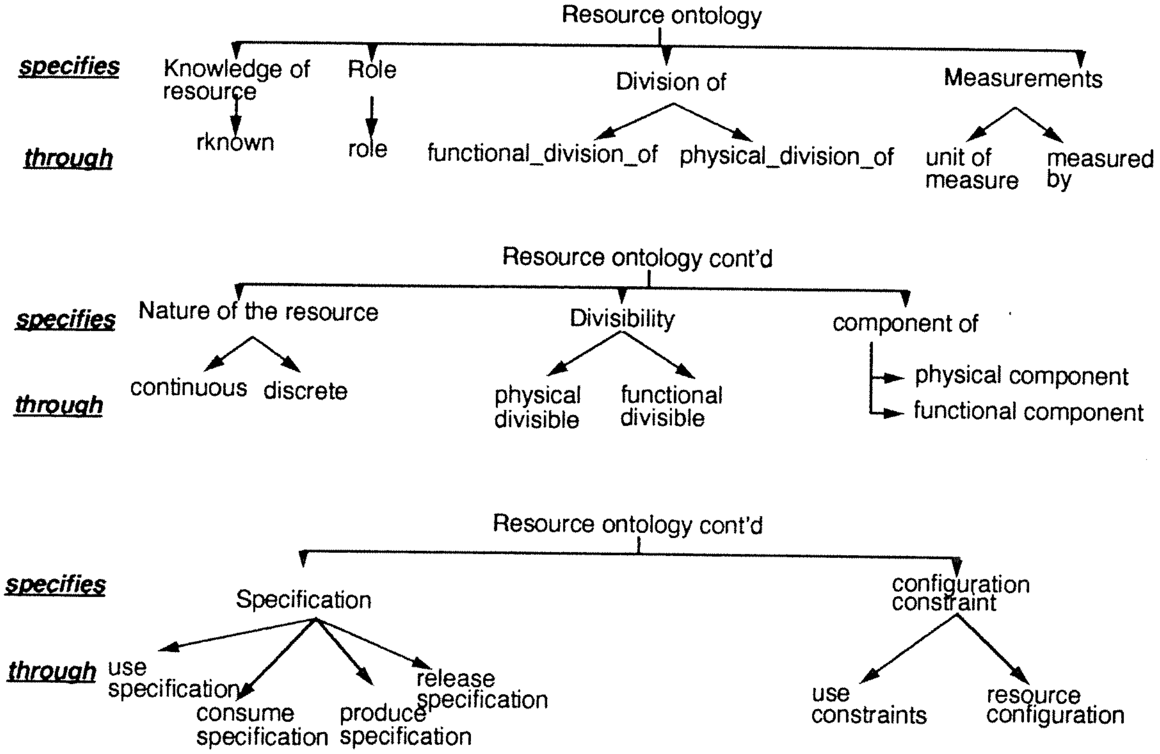
  A *time period* (pd)is a time construct that consists of two time points, representing the start and end time of an activity. A time interval/period also contains the minimum and maximin duration of the activity.
- **ST:** Starting time point of an activity.
- **ET:** End time point of an activity.
- **Dur:** Duration of the activity.
- **MinDur:** Minimum duration of the activity.
- **MaxDur:** Maximum duration of the activity.
- **Tp, tp:** *Tp* is a non instantiated time point while *tp* is instantiated to specific time point ID.
- **Ti, ti:** *Ti* is a non instantiated time period (interval) while *ti* is instantiated to specific time period ID.

4.2 Resource Ontology

What is presented in this section is the ontology for the representation of enterprise resources. The ontology is stratified in a sense that the definition of a term is based on the definition of previously defined resource (or activity-state) terms. The ontology first contains a number of assertions, also addressed as "ground terms", which form the basis or the core of the ontology. Examples of such terms are: knowledge of a resource (rknown), role, motility and division of each resource. Based on these terms more complex terms are defined such as: physical divisibility, physical component of, or the nature of the resource (continuos or discrete) a resource with respect to an activity. Finally, terms defining the capacity and the capacity trend of a resource are defined[1]. The resource is linked to the activity-state ontology as defined section 4.3.

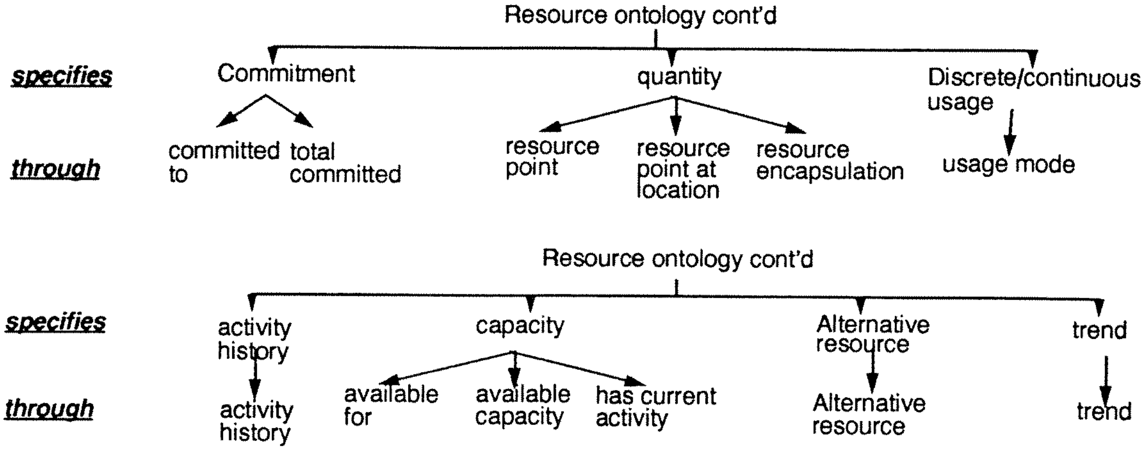
Terms are defined in first order logic, whenever possible, and implemented in Prolog. The FOL formulation will be printed in tilted characters and tagged with (FOL #) while the Prolog formulation will be printed in courier fonts and is tagged with (PRO #).

FIGURE 34 Resource ontology



[1] Figures 34 & 35 presents on overview of the ontology

FIGURE 35 Resource ontology cont'd



4.2.1 Resource-known: rknown(R)

Definition:

This is the most basic term in the ontology. It specifies *knowledge of a resource* as opposed to its physical existence. The importance of this definition lies in the fact that the ability to reason about a resource depends on it being known. This assumes that Prolog data base is consistent with the 'Rock' knowledge base.

Semantics and Implementation:

It is defined as a ground term. It is known if asserted in the Prolog database.

```
rknown(R).                                    (PRO 4)
```

If rknown is not asserted in Prolog data base, then the Rock knowledge base[1] is checked.

```
rknown(R):- kb_FRAME_ID(R).                   (PRO 5)
```

Example:

The following resources are asserted as being known for the use in the remainder of the chapter.

```
rknown(clip_base).                            (EX 1)
rknown(assembly_area_1).                      (EX 2)
```

[1] ROCK is a knowledge representation tool from Carnegie Group, Inc. Eventually all the knowledge base will be represented in ROCK and the axioms, in Prolog, will be calling ROCK functions, as shown in (PRO 5). However, throughout this chapter, it is assumed that the media of implementation is Prolog (Quintus Prolog).

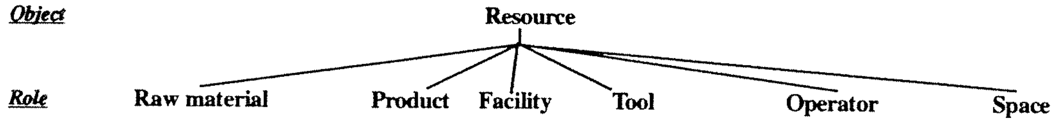
```
rknown(widget_No_1).                                  (EX 3)
rknown(short_arm).                                    (EX 4)
rknown(v_spring).                                     (EX 5)
rknown(wooden_chair).                                 (EX 6)
rknown(vehicle_1).                                    (EX 7)
```

4.2.2 Resource role: role(R, A, Role)

Definition:

According to webster a role is "an identifier attached to an index term to show functional relationships between terms". In TOVE, a resource has a role with respect to an activity.

FIGURE 36 Taxonomy of roles



Examples of roles include:

- **Raw material**: specify objects that are suitable for manufacture,
- **Product**: specifies that the role of the object with respect to an activity is as a manufactured object,
- **Facility**: specifies objects that facilitates an activity such as machines,
- **Tool**: specifies an instrument that is used or worked by hand,
- **Operator**: is the person who performs an activity,
- **Space**: specifies that a resource is a work area.

Semantics:

The term is defined as a tertiary ground term with three arguments:

- **R**: resource ID
- **A**: activity ID
- **Role**: is the argument denoting to the role of the resource with respect to an activity.

```
role(R, A, Role).                                     (PRO 6)
```

This entails that when a resource is defined as having a role with respect to an activity, then the resource can not have any other role with respect to the same activity.

$$(\forall\, r, a, role_1, role_2)\ role(r, a, role_1) \wedge role_1 \neq role_2 \supset \neg\, role(r, a, role_2) \qquad \text{(FOL 1)}$$

Example:

In figure 33, the role of the “short arm” resource with respect to “assemble clip reading lamp” activity is as a “**raw material**” while the role of the “assembly area 1” is a “**facility**”.

```
role(clip_reading_lamp, assemble_clip_reading_lamp,
product).                                                  (EX 8)
role(short_arm, assemble_clip_reading_lamp,
raw_material).                                             (EX 9)
role(assembly_area_1, assemble_clip_reading_lamp,
facility).                                                 (EX 10)
role(leg, fuel_a_fire, raw_material)                       (EX 11)
role(wooden_chair, fuel_a_fire, raw_material).             (EX 12)
role(multiplex_lines, facility).                           (EX 13)
role(clip_reading_lamp, assemble_clip_reading_lamp,
product).                                                  (EX 14)
role(clip_base, assemble_clip_reading_lamp,
raw_material).                                             (EX 15)
```

4.2.3 Motility:

Definition:

This notion specifies the ability of the resource to move, or be moved, from one position to another[1].

4.2.3.1 Mobile Resource:

Definition:

A resource is mobile if the activity requires it to be.

Semantics and Implementation:

It is defined as a ground term with the following arguments:

- R: resource ID,
- A: activity with which the resource is mobile

```
mobile(R, A)                                               (PRO 7)
```

Example:

[1] OPIS [Smith 89] has categorized a resource as being either a mobile or a stationary resource. A resource is stationary if it is either a cell-group or a work cell. A resource is mobile if the resource is either a human or a transport or a tool resource.

```
mobile(vehicle_1, transprt_1).
```
(EX 16)

4.2.3.2 Stationary Resource:

Definition:

A resource is stationary if the role of the resource with an activity requires it.

Semantics and Implementation:

It is defined as a ground term with the following arguments:

- R: resource ID,
- A: activity with which the resource is mobile

```
stationary(R, A)
```
(PRO 8)

Example:

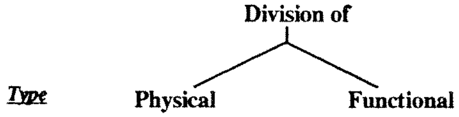
```
stationary(hammer_1, assemble_clip_reading_lamp).
```
(PRO 9)

4.2.4 Division of: ~_division_of(R2, R)

Definition:

This term specifies that a resource can be divided and one of the divisions is **R2**. There are two types of divisions: physical and functional.

FIGURE 37 Types of "division of"



"Physical division of" specifies a division that is neither mental, moral or imaginary but is related to the division of the body of an object; "functional division of" specifies a division affecting the function and not structure.[1]

Semantics and Implementation:

The "division of" terms are defined as ground terms with the following arguments:

- **R2**: the ID of the sub-division of the resource,
- **R**: the ID of the resource,
- **A**: the ID of the activity with which this classification is applicable.

As mentioned there are four "division of" declarations:

[1] implying that the resource is sharable

```
physical_division_of(R2, R, A).                    (PRO 10)
functional_division_of(R2, R, A).                  (PRO 11)
```

The implications of the division of terms are:

$$(\forall\, r_2, r, a)\, physical\_division\_of(r_2, r, a) \supset \neg functional\_division\_of(r_2, r, a) \quad \text{(FOL 2)}$$

Example:

```
functional_division_of(rectangle, oven).           (EX 17)
```

The above term specifies that the resource *oven* has a functional division of type *rectangle*. This term will be used in the capacity recognition process chapter five. A resource's capacity will be defined in terms of the number of its divisions. An oven's surface area could be defined in terms of a series of rectangular object with specific dimension. The *clip reading lamp* has a physical division which is *clip base*.

```
physical_division_of(clip_base, clip_reading_lamp).  (EX 18)
```

4.2.5 Divisibility of a resource: ~_divisible(R, A)

Definition:

This term specifies that a resource is divisible with respect to an activity without affecting its role. Similar to the delineation presented for the "division of" term, divisibility has three types: physical, functional and temporal divisibility.

A resource is physically divisible if the act of physically dividing the resource does not affect its role in the activity. In other words, the resource is physically divisible if each division can be used or consumed by an activity. That property is useful for planner/scheduler when deciding whether a portion of resource could support an activity. Functional divisibility of a resource, with respect to an activity, specifies that each division of the resource affects the functionality and not the structure of the resource. A "motor car" has a functional and physical division (e.g crank shaft) but the "motor" is neither functionally nor physically divisible with respect to "driving the car" activity. Finally, a resource is said to be temporal divisible if the use of a resource over time does not affect the future usability of the resource as in the case of the *multiplex lines* when associated with *communication activities*. This allows the system to reason about the nature of usage of the resource by an activity.

Semantics:

"A resource is physically divisible with respect to an activity if each physical division of the resource has the same role".

$$\forall(r, a)\, physical\_divisible(r, a) \equiv \forall(r_1, ro_1)\, rknown(r) \wedge physical\_division\_of(r_1, r) \wedge role(r_1, a, ro_1) \supset role(r, a, ro_1) \quad \text{(FOL 3)}$$

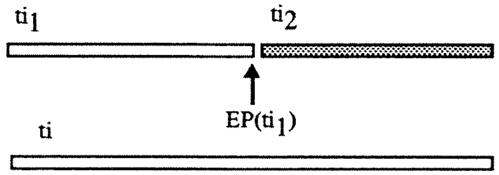
"A resource is functionally divisible with respect to an activity if each functional division of the resource has the same role".

$$\forall (r, a)\, \mathit{functional\_divisible}(r, a) \equiv$$

$$\forall (r_1, ro_1) \mathit{rknown}(r) \wedge \mathit{functional\_division\_of}(r_1, r) \wedge \mathit{role}(r_1, a, ro_1) \supset \mathit{role}(r, a, ro_1) \quad \text{(FOL 4)}$$

"A resource is temporally divisible with respect to an activity *A1* if there exists a time period in which two activities, including *A1*, were executing with the condition that the first activity (*A1*) was either suspended or completed and the resource had the same role with both activities. Moreover, both activities were not executing simultaneously (i.e overlapping constraint)."

FIGURE 38 Temporal divisibility - an example



$\forall (r, a)\ \mathit{temporal\_divisible}(r, a) \equiv \exists\ (ti, ti_1, ti_2, tp_1, tp_2, a_1, a_2, s_1, s_2)\ \mathit{rknown}(r) \wedge$

$(\mathit{uses}(s_1, r) \vee \mathit{consumes}(s_1, r)) \wedge (\mathit{uses}(s_2, r) \vee \mathit{consumes}(s_2, r)) \wedge$

$\mathit{is\_related}(a_1, s_1)$[1] $\wedge\ \mathit{is\_related}(a_2, s_2) \wedge$

$\mathit{time\_bound}(s_1, ti_1) \wedge \mathit{time\_bound}(s_2, ti_2) \wedge$

$\mathit{activity}(a_1, \mathit{executing}, tp_1) \wedge \mathit{period\_contains}(ti_1, tp_1) \wedge$

$((\mathit{activity}(a_1, \mathit{suspended}, tp\_end) \wedge tp\_end = EP(ti_1)) \vee$

$((\mathit{activity}(a_1, \mathit{completed}, tp\_end) \wedge tp\_end = EP(ti_1)) \wedge$

$\mathit{activity}(a_2, \mathit{executing}, tp_2) \wedge \mathit{period\_contains}(ti_2, tp_2) \wedge \neg\ \mathit{overlaps}(ti_1, ti_2) \wedge$

$\mathit{contains}(ti, ti_1) \wedge \mathit{contains}(ti, ti_2) \wedge \mathit{role}(r, a_1, \mathit{role}) \wedge \mathit{role}(r, a_2, \mathit{role}) \quad \text{(FOL 5)}$

Implementation:

```
physical_divisible(R, A):-
```

[1] is a term defined in the activity-state ontology that finds an activity is linked (related) to a state.

```
    rknown(R),
    physical_division_of(R1, R),
    role(R, A, Role), role(R1, A, Role).                    (PRO 12)
functional_divisible(R, A):-
    rknown(R),
    functional_division_of(R1, R),
    role(R, A, Role), role(R1, A, Role).                    (PRO 13)
temporal_divisible(R, A):-
    rknown(R),
    (uses(S1, R); consumes(S1, R));
    (uses(S2, R); consumes(S2, R));
    is_related(A1, S1), is_related(A2, S2),
    time_bound(S1, Ti1), time_bound(S2, ti2),
    activity(A1, executing, Tp1), period_contains(Ti1, Tp1),
    ((activity(A1, suspended, Tp_end), Tp_end = EP(Ti1));
    ((activity(A1, completed, Tp_end), Tp_end = EP(Ti1)),
    activity(A2, executing, Tp2), period_contains(Ti2, Tp2),
    contains(Ti, Ti1), contains(Ti, Ti2),
    \+overlaps(Ti1, Ti2),
    role(R, A1, ROLE), role(R, A2, ROLE).                   (PRO 14)
```

Example:

A wooden chair basically consists of:

- four legs.
- seat back part.
- seat part.

So, if the activity is to sit on the chair, accordingly the chair can not be considered physically divisible as the act of dividing the chair will result in the inability to perform the activity. While, on the other hand, if the activity is to *fuel_a_fire* by putting wood in the fire place, then the chair is considered physically divisible.[1]

[1] This is because there exist a physical division assertion, (EX 18), and the role of the division is the same as the "wooden chair" resource, (EX 12) & (EX 11).

```
?- physical_divisible(wooden_chair, fuel_a_fire).     (EX 19)
```

Moreover, the resource *short_arm* is not physically divisible for the activity *assemble_clip_reading_lamp* as it is not asserted that the resource has a physical division.

```
?- functional_divisible(oven, bake).                   (EX 20)
```

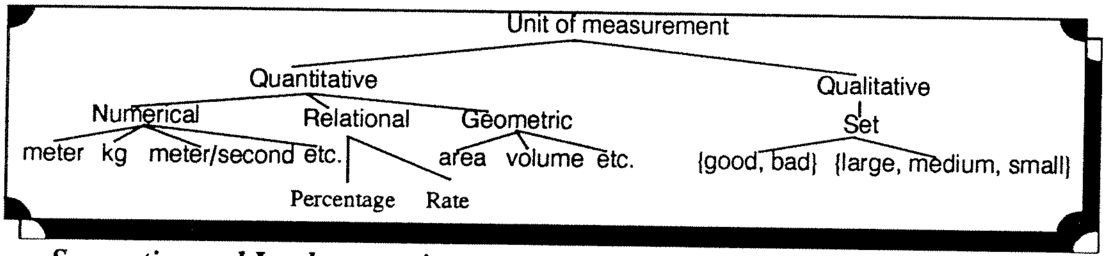
An *oven* is functionally divisible as the resource has a functional division with respect to *bake* activity and each division shares the same resource as that of the original resource (oven).

4.2.6 Unit of measurement: unit_of_measurement(R,UNIT ID, Unit, A)

Definition:

This predicate specifies a default measurement unit for a resource, when associated with activity. Accordingly, resource quantity or capacity is to be measured using the specified *unit* of measurement. This term is used for specifying both the qualitative and quantitative units of measurement. Qualitative units of measurement consist of an ordered set such as {large, medium small}. Qualitative units can be also used as a measure of quality: {good, bad}. Quantitative units are: numerical, geometric and relational units. Geometric units, used to represent area/volume, are for example used in the capacity recognition of a resource[1]. The relational units are units the specify the rate or percentage of usage. Quantitative units are used to specify attributes such as weight, length, capacity[2].

FIGURE 39 Taxonomy of unit of measurement



Semantics and Implementation:

In this context the predicate is defined as a ground term with three functional parameters:

- **R:** is the resource ID.

[1] described in the capacity chapter

[2] KRSL project [Allen et al 92] includes a similar delineation of unit of measurement. KRSL has defined unit-types as either being qualitative or quantitative. Chapter three contains more information concerning the subject.

- **Unit-ID**: the ID of the unit of measurement. It specifies for example whether the weight, length, volume (etc.) is going to be measured.
- **Unit:** is the measurement unit.
- **A:** is the ID of the activity with which the unit of measurement is associated.

```
unit_of_measurement(R, Unit_ID, Unit, A).                (PRO 15)
```

The parameter *Unit* could be instantiated to[1]:

- a **weight** measuring unit - *ton,*
- **length** measuring unit - *kilometer,*
- **capacity** measuring unit, as described in the capacity recognition chapter.

A constraint on the "unit of measurement" is that there should be a corresponding "measured_by" assertion[2].

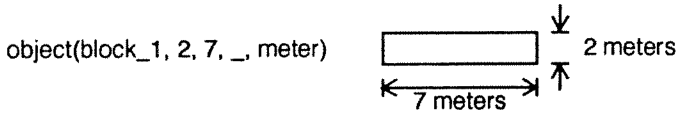
Example:

```
unit_of_measurement(sand, weight, kilogram,
concrete_mix).                                            (EX 21)
```

This assertion states that the *weight* of the object *sand* is measured in terms of kilograms when the activity is *concrete mix.* Consider the case of defining the unit of measurement for the capacity recognition process. The resource *oven* has *block 1* as its unit of measure.

```
unit_of_measurement(oven_1, capacity, block_1,
bake_small_pizza).                                        (EX 22)
```

FIGURE 40 unit of measurement - block(2, 7, _, meters)



As for the qualitative unit of measurement, the unit of measurement is represented as an ordered list with fixed number and types of elements.

```
unit_of_measurement(runway, quality, qualitative,
plane_landing).                                           (EX 23)
```

[1] These units are predefined units

[2] defined in next section

A *runway* has *qualitative* as a unit of measure[1].

4.2.7 Measured by: measured_by(R, Unit_ID, A).

Definition:

"Measured by" defines the objects by which a resource is measured with respect to an activity. This term acts as a constraint on the "unit of measurement" term. Each unit of measure must have a corresponding "measured by" assertion. For example, a runway is measured in terms of runway-length and runway-quality[2].

Semantics and Implementation:

Defined as a ground term with the following parameters:

- **R**: resource ID,
- **Unit**: unit ID of the measurement unit[3],
- **A**: the activity associated with the units of measurement.

```
measured_by(R, Unit_id, A).                                    (PRO 16)
```

As mentioned constraint on the "unit of measurement" is that there should be a corresponding measured by assertion:

$$(\forall\ r,\ unit\_id,\ a)\ measured\_by(r,\ unit\_id,\ a) \supset (\exists\ u)\ unit\_of\_measurement(r,\ unit\_id,\ u,\ a) \qquad \text{(FOL 6)}$$

Example:

```
measured_by(runway, length, plane_landing).                    (EX 24)
measured_by(runway, quality, plane_landing).                   (EX 25)
```

The resource *runway* is measured in terms of its length and quality when the resource is associated with *plane landing* activities. The "measured by" assertion should be coupled with the assertions of "unit of measurement". That is to say to couple with:

```
unit_of_measurement(runway, length, meter,
plane_landing).                                                (EX 26)
```

[1] object(qualitative, [accepted, not_accepted],_,_).

[2] KRSL project [Allen et al 92] has also defined the concept of measured by.

[3] a constant

4.2.8 Resource requirements specification for activities

The predicates in this section are defined for notational convenience. The consumption, use, produce and release specifications specify the amounts of the resource to be consumed, used, produced or released respectively over a time interval, as well as the unit of measurement. The information included in these specifications[1] are already defined in the activity state ontology.

4.2.8.1 Consumption specification: consumption_spec(R, A, Ti, Q, Rate, Unit)

Definition:

This predicate specifies for an activity the amount of a resource is to be consumed during a specified time interval and unit of measurement. The amount specified in this term is the total amount that is to be consumed after the completion of an activity. That is to say, if the amount of consumption is in terms of rate the amount would state the total amounts consumed.

Semantics and Implementation:

The predicate is defined as a ground term with the following arguments:

- **R**: the resource ID.
- **A**: activity using the resource.
- **Ti**: time interval ID of the activity.
- **Q**: the total amount required to be consumed by the activity.
- **Rate:** is used to define whether the resource will be consumed on continuos or discrete basis[2].
- **Unit**: unit of measurement.

```
consumption_spec(R, A, Ti, Q, Rate, Unit).
```
(PRO 17)

The consumption specification concatenates the specification into one term as defined in the following FOL formulation:

$$(\exists\, a, q, ti, rate, unit)\ (\forall r)\ \mathit{consumption\_spec}(r, a, ti, q, rate, unit) \equiv (\exists\, s, s_2, \mathit{unit\_id})$$

$$\mathit{enabling}(s, a) \wedge \mathit{is\_related}(a, s_2) \wedge \mathit{consumes}(s_2, r) \wedge \mathit{quantity}(s_2, q) \wedge \mathit{time\_bound}(s_2, ti) \wedge$$

$$\mathit{unit\_of\_measurement}(r, \mathit{unit\_id}, unit, a) \wedge \mathit{measured\_by}(r, \mathit{unit\_id}, a) \qquad \text{(FOL 7)}$$

"The consumption specification term entails that the resource amount will be decremented after the completion of the activity". [3]

[1] i.e the arguments of each term

[2] defined in the *Usage mode* section 4.2.14.

$$(\forall r, a, s, ti, q', rate, unit)\ consumption\_spec(r, a, ti, q', rate, unit) \equiv (\forall\ s, r, a, q, ti, tp, tp')$$
$$rp(r, q, tp, unit\_id) \wedge ((is\_related(a, s) \wedge consumes(s, r)) \wedge$$
$$tp = SP(ti) \wedge tp' = EP(ti) \wedge enabling\_state(s, tp, enabled)$$
$$\supset rp(r, q - q', tp', unit) \qquad \text{(FOL 8)}$$

Example:

```
consumption_spec(clip_base,assemble_clip_reading_lamp,
pd1, 1, 1, object).                                   (EX 27)
```

The term specifies that activity *assemble_clip_reading_lamp* requires one *clip_base*, objects, resource over time interval *pd1* to be consumed.

4.2.8.2 Use specification: use_spec(R, A, Ti, Q, Rate, Unit)

Definition:

This predicate specifies the requirement of an activity to use a resource at a specified time.

Semantics and Implementation:

Semantically, the predicate is defined as a ground term with parameters:

- **R**: the resource ID.
- **A**: activity using the resource.
- **Ti**: the time interval ID of the activity.
- **Q**: specifies the portion of the resource to be used.
- **Rate:** is used to define whether the resource will be used on continuos or discrete basis[1].
- **Unit**: Unit of measurement

```
use_spec(R, A, Ti, Q, Rate, Unit).                    (PRO 18)
```

The use specification concatenates the specification into one term as defined in the following FOL formulation:

$$(\exists\ a, q, ti, rate, unit)\ (\forall r)\ use\_spec(r, a, ti, q, rate, unit) \equiv (\exists\ s, s_2, unit\_id)\ enabling(s, a) \wedge$$
$$is\_related(a, s_2) \wedge uses(s_2, r) \wedge quantity(s_2, q) \wedge time\_bound(s_2, ti) \wedge$$
$$unit\_of\_measurement(r, unit\_id, unit, a) \wedge measured\_by(r, unit\_id, a) \qquad \text{(FOL 9)}$$

[3] In TOVE, this axiom is called an effect axioms which links the resource ontology with the causal theory of activity [TOVE 94].

[1] defined in the *Usage mode* section 4.2.14.

"The use specification term entails that the resource amount will remain constant, if the resource is being used" [1]

$$(\forall r, a, s, ti, q', rate, unit)\ use\_spec(r, a, s, ti, q', rate, unit) \equiv (\forall s, r, a, q, ti, tp, tp')$$

$$rp(r, q, tp, unit\_id) \wedge (is\_related(a, s) \wedge uses(s, r)) \wedge$$

$$tp = SP(ti) \wedge tp' = EP(ti) \wedge enabling\_state(s, tp, enabled)$$

$$\supset rp(r, q, tp', unit) \qquad \text{(FOL 10)}$$

Example:

```
use_spec(assembly_area_1, assemble_clip_reading_clamp,
pd1, 90, 90, block_1).
```
(EX 28)

The above term specifies that *assemble_clip_reading_clamp* uses *assembly_area_1* resource over time interval *pd1*. The activity will occupy 90 block_1[2].

```
use_spec(assembly_area_1, assemble_hand, pd1, 30, 30,
block_1).
```
(EX 29)

The above term specifies that *assemble_hand* uses the *assembly_area_1* resource over time interval *pd1* with a portion of usage of thirty units of capacity.

```
use_spec(hammer_1, assemble_hand, pd1, 1, 1, object).
```
(EX 30)

```
use_spec(hammer_1, assemble_base, pd2, 1, 1,
object).
```
(EX 31)

```
use_spec(assembly_area_2, assemble_base, pd2, 1, 1,
object).
```
(EX 32)

4.2.8.3 Production specification: produce_spec(R, A, Ti, Q, Rate, Unit)

Definition:

A *produce* state "signifies that a resource which did not exist prior to the performance of an activity, has been created by the activity" [Tove 92]. 'produce spec', which is a result of a scheduling/planning activities, specifies that amount that is to be produced by an activity. Moreover the term specifies when the production is going to be achieved.

Semantics and Implementation:

[1] In TOVE, this axiom is called an effect axioms which links the resource ontology with the causal theory of activity [TOVE 94].

[2] object(block_1, 10, 10, _, cm) is the unit of measurement and it is a capacity unit of measurement.

Similar to use and consume specification the "produce spec" term is defined as ground term with six arguments:

- **R**: ID of the produced resource.
- **A**: ID of the activity producing the resource.
- **Ti**: time period at the end of which the production activity will be over.
- **Q**: total quantity produced after the completion of the activity.
- **Rate:** is used to define whether the resource will be produced on continuos or discrete basis[1].
- **Unit**: unit of measurement of the resource.

```
produce_spec(R, A, Ti, Q, Rate, Unit).
```
(PRO 19)

The produce specification concatenates the specification into one term as defined in the following FOL formulation:

$$(\exists\, a, q, ti, rate, unit)\ (\forall r)\ produce\_spec(r, a, ti, q, rate, unit) \equiv (\exists\, s, s_2, unit\_id)\ causes(s, a) \wedge produces(s_2, r) \wedge is\_related(a, s_2) \wedge quantity(s, q) \wedge time\_bound(s, ti) \wedge unit\_of\_measurement(r, unit\_id, unit, a) \wedge measured\_by(r, unit\_id, a) \quad \text{(FOL 11)}$$

"The produce specification term entails that the resource amount will increase by a constant, if the resource is being used" [2]

$$(\forall r, a, s, ti, q', rate, unit)\ use\_spec(r, a, s, ti, q', rate, unit) \equiv (\forall\, s, r, a, q, ti, tp, tp')\ rp(r, q, tp, unit\_id) \wedge (is\_related(a, s) \wedge produces(s, r)) \wedge tp = SP(ti) \wedge tp' = EP(ti) \wedge enabling\_state(s, tp, enabled) \supset rp(r, q + q', tp', unit) \quad \text{(FOL 12)}$$

Example:

```
produce_spec(short_arm, assemble_short_arm, pd3, 1, 1,
object).
```
(EX 33)

The above specifies that the "assemble short arm" is producing one unit of "short arm" resource at the end time of pd1 time period.

[1] defined in the *Usage mode* section 4.2.14.

[2] In TOVE, this axiom is called an effect axioms which links the resource ontology with the causal theory of activity [TOVE 94].

4.2.8.4 Release specification: release_specification(R, A, Ti, Q, Rate, Unit)

Definition:

A *release* state "signifies that a resource, which has been designated as being used is now available for use/consumption elsewhere" [TOVE 92]. 'Release spec', which is a result of a scheduling/planning activities, specifies that amount that is to be released by an activity. Moreover the term specifies when the release action is going to be achieved.

Semantics and Implementation:

Similar to use and consume specification the "release spec" term is defined as ground term with five arguments:

- **R**: ID of the released resource.
- **A**: ID of the activity releasing the resource.
- **Ti**: time period at the end of which the release will be achieved.
- **Q**: total quantity released.
- **Rate:** is used to define whether the resource will be released on continuos or discrete basis[1].
- **Unit**: unit of measurement of the resource.

```
release_spec(R, A, Ti, Q, Rate, Unit).
```
(PRO 20)

The release specification concatenates the specification into one term as defined in the following FOL formulation:

$$(\exists\, a, q, ti, rate, unit)\ (\forall r)\ release\_spec(r, a, ti, q, rate, unit) \equiv (\exists\, s, s_2, unit\_id)\ causes(s, a) \wedge releases(s_2, r) \wedge is\_related(a, s_2) \wedge quantity(s, q) \wedge time\_bound(s, ti) \wedge unit\_of\_measurement(r, unit\_id, unit, a) \wedge measured\_by(r, unit\_id, a) \qquad \text{(FOL 13)}$$

A constraint on the release specification is the quantity parameter (Q) defined in the use and release specification are equal. Moreover the starting time of the time interval in the release specification should be after the starting time of the starting of the time interval defined in the use specification.

$$(\exists\, a, q, ti, ti_2, rate\_1, rate\_2, unit)\ (\forall r)\ release\_spec(r, a, ti, q, rate\_1, unit) \supset (use\_spec(r, a, ti_2, q, rate\_2, unit) \wedge before(SP(ti_2), SP(ti)) \qquad \text{(FOL 14)}$$

Example:

[1] defined in the *Usage mode* section 4.2.14.

```
release_spec(assembly_area_2, assemble_base, pd3, 1, 1,
object).                                                    (EX 34)
```

The above specifies that the *assembly area 2* resource will be releases by *assemble base* activity at the end time of pd3 time period.

4.2.9 Continuous vs. discrete resource

Definition:

A continuous resource indicates a resource whose physical divisions are uncountable. These resources are marked by uninterrupted extension in volume. Discrete resources on the other hand specify that a resource is a countable one. Both terms are defined relative to an activity[1].

Semantics:

Both terms are defined as a ground term with two arguments:

- **R**: ID of the resource
- **A**: ID of the activity with which the resource is either continuous or discrete.

$$(\forall\, r, a)\; continuous(r, a) \equiv physical\_divisible(r, a) \qquad \text{(FOL 15)}$$

$$(\forall\, r, a)\; discrete(r, a) \equiv \neg\, continuous(r, a) \qquad \text{(FOL 16)}$$

If the resource is discrete then the consumption or the use specification is in terms of integer amounts[2].

$$(consumption\_spec(r, a, ti, q, rate, u) \lor use\_spec(r, a, ti, q, rate, u)) \land discrete(r, a) \supset integer(q) \qquad \text{(FOL 17)}$$

Implementation:

```
continuous(R, A):-

   physical_divisible(R, A).                                (PRO 21)

discrete(R, A):-

   \+ continuous(R, A).                                     (PRO 22)
```

Example:

```
   continuous(water, drinking).                             (EX 35)
```

[1] Hayes [Hayes 90] has presented an ontology for liquid in the context of *naive physics* defined in first order logic. Liquid was chosen because it has no definite shape, merge and split in mysterious ways. Hayes specify that "spatio-temporal continuity is the criterion of determining the identity of complex assemblies". Included in the endeavor is ontology on describing geometry, change, shape and time.

[2] integer(q) is used specifying that q is an integer

```
discrete(clip_base, assemble_clip_base).                    (EX 36)
discrete(short_arm, assemble_clip_readin_lamp).             (EX 37)
```

4.2.10 Component_of: ~_component_of(R1, R2, A, Type)

Definition:

"Component of" specifies a resource as being a sub division of another resource and that the division does not share the same role with the original resource. A resource can be a physical or functional component of another resource with respect to an activity. This term is used for example in the bills of material explosion of parts[1].

Semantics:

"A resource *R2* is a physical component of resource *R1* if *R2* is a physical division of the *R1* and both resources do not share the same role with respect to an activity[2]".

$$(\forall r_1, r_2)\, physical\_component\_of(r_2, r_1, a) \equiv \forall(r, r_2, ro_1)\, physical\_division\_of(r_2, r_1) \wedge role(r_2, a, ro_1) \wedge \neg role(r_1, a, ro_1). \quad \text{(FOL 18)}$$

"A resource *R2* is a functional component of resource *R*1 if *R2* is a functional division of the *R1* and both resources do not share the same role with respect to an activity".

$$(\forall r_1, r_2)\, functional\_component\_of(r_2, r_1, a) \equiv \forall(r, r_2, ro_1)\, functional\_division\_of(r_2, r_1) \wedge role(r_2, a, ro_1) \wedge \neg role(r_1, a, ro_1). \quad \text{(FOL 19)}$$

Implementation:

```
component_of(R2, R1, A, Type):-
    rknown(R),
    physical_component_of(R2, R1, A, Type);
    functional_component_of(R2, R1, A, Type).               (PRO 23)
physical_component_of(R2, R1, A, Type):-
    rknown(R), physical_division_of(R2, R1),
```

[1] OPIS [Smith 89] includes a similar definition equivalent to *physical component of*by using the *sub-resource* relation. This relation is the basis of the hierarchial resource description that is used in OPIS modelling framework.

[2] But the division has a role never the less with respect to the activity.

```
        role(R2, A, Role2),
        \+role(R1, A, Role2),
        Type = physical.                                   (PRO 24)
    functional_component_of(R2, R1, A, Type):-
        rknown(R), functional_division_of(R2, R1),
        role(R2, A, Role2),
        \+role(R1, A, Role2),
        Type = functional.                                 (PRO 25)
```

Example:

The clip reading lamp consists of three components:

- clip_base.
- short_arm.
- small_head.

```
physical_division_of(clip_base, clip_reading_lamp).  (EX 38)
physical_division_of(short_arm, clip_reading_lamp).  (EX 39)
physical_division_of(small_head, clip_reading_lamp). (EX 40)
```

Furthermore, *clip_base* resource consists of the following parts:

- v_spring.
- round_nut.
- bolt_4
- v_pad

```
physical_division_of(v_spring, clip_base).                (EX 41)
physical_division_of(round_nut, clip_base).               (EX 42)
physical_division_of(bolt_4, clip_base).                  (EX 43)
physical_division_of(v_pad, clip_base).                   (EX 44)
```

Now, based on the above physical division assertions, if the following query is checked:

```
?- component_of(short_arm, clip_reading_lamp,
assemble_clip_reading_lamp, Type).                        (EX 45)
```

the variable *Type* having the value of **physical** as *short arm* has a physical division of *clip reading lamp* but they don't share the same role, (EX 8) & (EX 9).

```
?-component_of(crank_shaft,motor,driving_a_car,Type).(EX 46)
```

the variable *Type* having the value of **functional & physical** as *crank shaft* has a "functional and physical division of" *motor* but the division does not share the same role with that of motor.

4.2.11 Resource point:

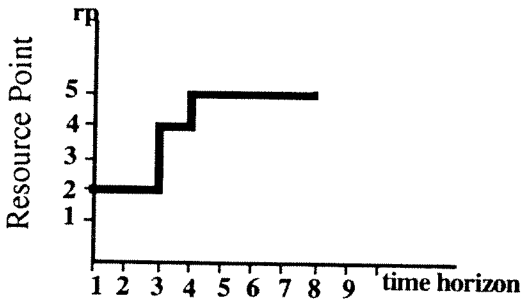
Reasoning about activities and processes requires a good representation of quantities, because processes/activities generally are initiated and terminated when the value or ordering of quantities change [Forbus 84]. Quantities are specified through the use of the **resource point** predicate. This predicate is used to represent the resource's quantity at a certain point of time. Resource point specifies the amount of a resource that *physically* exists. The specified amount may or may not be committed to activities[1].

4.2.11.1 Resource point at Time T: rp(R, Q, Tp, U)

Definition:

A resource point is a predicate specifying the resource quantity at a specific time point. Resource points are defined with respect to a specific unit of measurement.

FIGURE 41 resource point at a specific time point



Semantics and Implementation:

rp defines that resource R exists in quantity Q at time point Tp in units U. Resource point is a ground term with three arguments:

- **R**: resource ID.
- **Q**: quantity at the specified time point.
- **Tp**: Time point ID at which check is done.
- U: is the unit of measurement.

```
rp(R, Q, Tp, U).
```
(PRO 26)

[1] In qualitative (naive) physics domain, Forbus [Forbus 84] uses *(M Q t)* which defines the value Q at time t.

Example:

```
rp(v_spring, 100, tp3, object).                    (EX 47)
```

This predicate states that there exists a resource point, for resource *v_spring* at time point *tp3*, with quantity of 100 objects.

4.2.11.2 Resource point at Location L at time Tp: rpl(R, Q, Tp, L, Unit)

Definition:

Resource point is now extended to differentiate quantities at different locations.

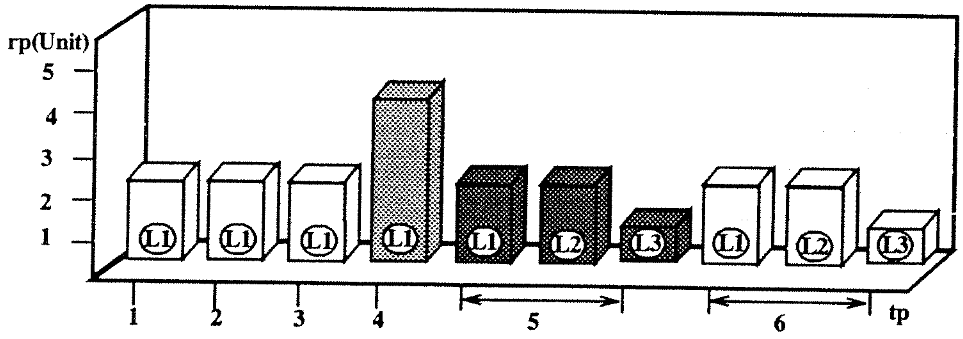
Semantics:

It is defined as a ground term with the following parameters:

- **R**: resource ID.
- **Q**: quantity.
- **Tp**: time point ID.
- **L**: location's ID of the resource.
- **Unit**: is the unit of measurement.

```
rpl(R, Q, Tp, L, Unit).                    (PRO 27)
```

FIGURE 42 resource point at a time point and location



Now with this definition of the resource point at a location and time - rpl(R, Q, T, L, U), resource point at time T - rp(R, Q, Tp, U) could be further defined as the summation of all resource points of a resource at time point (Tp) over all locations[1].

[1] i.e for a fixed set of locations

$$\begin{gathered}(\forall\ r)(\exists\ Q, tp, unit)\ rp(r, Q, tp, unit) \equiv (\exists\ q1, q2, q3 \ldots\ldots\ldots qn, sl1, sl2, \ldots\ldots\ldots\ sln) \\ rknown(r) \wedge rpl(r, q1, tp, s11, unit) \wedge rpl(r, q2, tp, s12, unit) \wedge rpl(r, q3, tp, s13, unit) \\ \wedge \ldots\ldots\ldots rpl(r, qn, tp, s1n, unit) \wedge Q = q1 + q2 + q3 \ldots\ldots\ldots\ qn \qquad \text{(FOL 20)}\end{gathered}$$

Implementation:

In Prolog, the definition is:

```
rp(R, Q, Tp, Unit):-
    rknown(R),unit_of_measurement(R, Unit_id, Unit, A),
    measured_by(R, Unit_id, A),
    findall(QQ, rpl(R, QQ, Tp, L, U), List),
    sum_rp(List, 0, QQ).                                    (PRO 28)
sum_rp(List, Q, Q).

sum_var([H|T], Q1, Q):-
    Q2 is Q1 + H,
    sum_rp(T, Q2, Q).                                       (PRO 29)
```

When rp axiom is used, the total quantity of resource *R* will be returned, in terms of the specified unit of measurement, at a time point *Tp* over all locations.

Example:

First, the rpl(R, Q, Tp, L, Unit) predicates are defined.

```
rpl(clip_base, 4, tp1, s2, object).                         (EX 48)
rpl(v_spring, 100, tp3, s1, object).                        (EX 49)
rpl(v_spring, 125, tp3, s2, object).                        (EX 50)
rpl(v_spring, 10, tp6, s3, object).                         (EX 51)
```

The last ground term, for example, specifies that the resource point of the resource *v_spring* is *10 objects* at time point *tp6* in location *s3*. Now, rp(R, Q, Tp, U) is used to get the total quantity of the resource *v_spring* at a specific time point by using:

```
?- rp(v_spring, Q, tp3, object).                            (EX 52)
```

which will return the total quantity of the resource *v_spring* at time point *tp3* which is equal to *225*. This done by satisfying the goals rp(R, Q, Tp, U) axiom((PRO 28) using rpl(R, Q, Tp, L, U) assertions ((EX 49),(EX 50)).

4.2.12 Encapsulation of resource points

Definition:

Resource point encapsulation transforms a resource point from one unit of measurement to another. Encapsulation in general is a transformation of a application specific data into a form that is in agreement to a common knowledge representation [Gruber et al 92]. Sometimes different activities require different units of measurement of the same resource. Accordingly, whenever a resource point is required in a specific unit of measurement, and a resource point is asserted in a different unit of measurement, resource point encapsulation is to be used[1].

Semantics:

"A resource has a resource point in terms of Unit1 if there exists Unit2 for which rp(R, Qq, Tp, Unit2) and transformation(Unit1, Unit2, R) exist".

$$(\forall r)\ (\exists q, tp, unit1)\ rp(r, q, tp, unit1) \equiv (\exists unit2, q2)\ rp(r, q2, tp, unit2) \wedge transformation(q2, q, unit1, unit2, r)) \qquad \text{(FOL 21)}$$

Implementation:

```
rp(R, Q, Tp, Unit1):-

   rknown(R),

   ((\+measured_by(R, unit_id, unit1, A),

   Q = can_not_measure_with_this_unit);

   measured_by(R, unit_id, unit1, A)

   rpl(R, Q2, Tp, L, Unit2),

   transformation(Q2, Q, Unit1, Unit2, R).                (PRO 30)

transformation(Q2, Q, Unit1, Unit2, R):-

   (((Unit2 == object, Unit1 == pair_object[2]), Q is Qq/2);

   ((Unit2 == object, Unit1 == lot), Q is Qq/10))         (PRO 31)
```

Example:

Assuming the assertion of the resource point of *air_port_run_way* resource is *1000 meter.* How ever if the needed resource point is in different unit of measure, *lot* for example, then using:

```
?- rp(airport_run_way, Q, tp6, cm).                       (EX 53)
```

[1] KRSL [Allen et al 92] uses *unit relation* as the means of converting between unit of measurements.

[2] These transformation are defined just for the sake of clarification.

would return the variable *Q* instantiated to *1000,00 (cm)* after performing the transformation process.

4.2.13 Resource exist:

We define the predicate resource-exist (**rexist**) as specifying the physical existence of a resource. That is to say, this term does neither specify the knowledge of a resource nor the existence of the "notion/concept" of the resource/object. The predicate is used to specify/check the physical existence of the resource in the past, present and future. A resource could physically exist but that does not indicate that it could be used or consumed by an activity. The existence predicate only specifies that the resource could be physically located. Again the predicate is defined at two levels[1].

4.2.13.1 Resource exist at time t: rexist(R, Tp)

Definition:

"rexist" is true if the quantity of the resource at time Tp is greater than zero.

Semantics:

The resource has to be known and the resource has to have a resource point with quantity greater than zero at a specified time point.

$$(\forall r)\ (\exists tp)\ rexist(r, tp) \equiv rknown(r) \wedge (\exists l, q, u)\ rpl(r, q, tp, l, u) \wedge (q > 0) \quad \text{(FOL 22)}$$

Implementation:

The predicate is defined having two arguments:

- **R**: resource ID.

```
Tp: time point ID at which existence is checked.

rexist(R, Tp):-

   rknown(R), rpl(R, Q, Tp, L, U), (Q > 0).          (PRO 32)
```

When the resource point is used to check the existence of a resource, both the location and the unit arguments in the resource point predicate are neglected[2].

Example:

[1] Hirst [Hirst 89] on the other hand defines existence as being both physical and non-physical. "Everything exists". This includes specifying the existence of the notion "a squared circle".

[2] defined as anonymous variables.

If we need to check the physical existence of the resource *v_spring*, at a specific time point *tp3*, the resource *v_spring* should be known and the resource should have a resource point with quantity which is greater than zero.

```
        ?- rexist(v_spring, tp3).                                  (EX 54)
```

The use of the above predicate would return **Yes** as an answer because the goals of rexist axiom were satisfied by the ground term assertions of *rknown(v_spring)* and *rpl(R, Q, Tp, L, Unit)* ((EX 49),(EX 50)).

However if the following predicate is used to check the physical existence of the resource *v_spring* at time point *tp5*:

```
        ?- rexist(v_spring, tp5).                                  (EX 55)
```

No would be returned as the second goal of rexist axiom could not be satisfied as no resource point for the resource *v_spring* is asserted at time point *tp5*.

4.2.13.2 Resource exist in a location: rexistl(R, Tp, L)

Definition:

This predicate specifies the physical existence of the resource at a specific location at a specific time.

Semantics:

For a resource to exist in a location at a specific time, the resource should be known, should have a resource point (rpl(R, Q, Tp, L, U)) and a quantity which is greater than zero.

$$(\forall\, r)(\exists\, tp, l)\ rexistl(r, tp, l) \equiv rknown(r) \wedge (\exists\, q, u)\ rpl(r, q, tp, l, u) \wedge (q > 0) \qquad \text{(FOL 23)}$$

Implementation:

The term is defined with three arguments:

```
    rexistl(R, Tp, L):-

        rknown(R), rpl(R, Q, Tp, L, Unit), (Q > 0).               (PRO 33)
```

Example:

If we need to check the physical existence of the resource *v_spring*, at a specific time point and place, the resource *v_spring* should be known and the resource should have a resource point - rpl(R, Q, T, L, Unit) with quantity greater than zero.

```
        ?- rexistl(v_spring, tp3, L).                              (EX 56)
```

The use of the above predicate will return the variable *L* equal to *s1* and *s2* indicating the physical existence of the resource *v_spring* at locations *s1* and *s2*. This is done by satisfying goals of rexistl axiom by the use of the ground term assertions of *rknown(v_spring)* and *rpl(R, Q, Tp, L, Unit)* ((EX 49),(EX 50)).

4.2.14 Usage Mode: usage_mode(R, A, Result)

Definition:

Usage mode axiom returns whether a resource supports an activity on a discrete or continuous basis. The term does not imply that the activity is discrete or continuous. The mode of usage is depended on the activity that uses/consumes the resource.

Semantics:

"The mode of usage is achieved through checking the use/consume/produce specification term. If the quantity term *(Q)* is equal to the rate term *(Rate)* in the specification, then the process is discrete otherwise the process is continuous with a rate which is equal to the *rate* parameter"

$$(\forall\, r, a)\ \mathit{continuous\_mode}(r, a) \equiv (\exists\, q, \mathit{unit}, \mathit{rate}, \mathit{ti})\ (\mathit{rknown}(r)$$

$$(\mathit{use\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \vee \mathit{consumption\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \vee$$

$$\mathit{produce\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit})) \wedge q \neq \mathit{rate} \qquad \text{(FOL 24)}$$

$$(\forall\, r, a)\ \mathit{discrete\_mode}(r, a) \equiv (\exists\, q, u, \mathit{rate}, \mathit{unit})\ ($$

$$(\mathit{use\_spec}(r, a, \mathit{ti}, q, q, \mathit{unit}) \vee \mathit{consumption\_spec}(r, a, \mathit{ti}, q, q, \mathit{unit}) \vee$$

$$\mathit{produce\_spec}(r, a, \mathit{ti}, q, q, \mathit{unit})) \qquad \text{(FOL 25)}$$

If a usage mode of a resource is continuous with respect to an activity, that implies that the resource is continuous.

$$(\forall\, r, a)\ \mathit{continuous\_mode}(r, a) \equiv \mathit{continuous}(r, a) \qquad \text{(FOL 26)}$$

Implementation:

```
usage_mode(R, A, Result):-
    rknown(R),
    continuous(R, A, Result)|
    discrete(R, A, Result).                                  (PRO 34)
continuos_mode(R, A, Result):-
```

```
    (use_spec(R, A, Ti, Q, Rate, U);
    consumption_spec(R, A, Ti, Q, Rate, U);
    produce_spec(R, A, Ti, Q, Rate, U)),
    Q \== Rate,
    Result = continuous.                                  (PRO 35)


discrete_mode(R, A, Result):-
    (use_spec(R, A, Ti, Q, Q, U);
    consumption_spec(R, A, Ti, Q, Q, U);
    produce_spec(R, A, Ti, Q, Q, U)),
    Result = discrete.                                    (PRO 36)
```

Example:

```
    ?- usage_mode(assembly_area_2, assemble_base,
    Result).                                              (EX 57)
```

use_spec(assembly_area_2, assemble_base, pd2, 1, 1, object)

The use of the usage_mode axiom, with the resource *assembly_area_2* and activity *assemble_*base, would return **Result** = **discrete**. This is because the *Q* and *Rate* parameters are equal (EX 28).

4.2.15 Simultaneous Use Restriction: simultaneous_use_restriction(A1, A2, R)

Definition:

Simultaneous use restriction prohibits the use/consumption of a resource by two activities simultaneously. For example when two activities require the same oven but at different temperatures or because one activity negatively interacts with another[1].

Semantics and Implementation:

The predicate is defined as ground term with the following parameters:

- **A1**: activity ID of the first activity using a resource.
- **A2**: activity ID of the second activity using a resource.

[1] A *reusable* resource in SIPE [Wilkins 88] is defined as being able to support only one activity at a time, so each activity has a simultaneous use restriction with any other when requiring the same resource.

- **R**: ID of the resource to be used.

```
simultaneous_use_restriction(A1, A2, R).
```
(PRO 37)

This term specifies that activities *A1* and *A2* can not be supported by resource *R* at the same time. Accordingly, entailing that both activities can not commit the resource over two overlapping intervals.

$$(\forall\, a1, a2, r)\ \mathit{simultaneous\_use\_restriction}(a1, a2, r) \equiv (\forall\, s, s2, r, a, a2)\ (\neg\, \exists tp)\ \mathit{use}(s, a) \wedge \mathit{uses}(s, r) \wedge \mathit{use}(s2, a2) \wedge \mathit{uses}(s2, r)$$

$$\supset \mathit{enabling\_state}(s, tp, \mathit{enabled}) \wedge \mathit{enabling\_state}(s_2, tp, \mathit{enabled}) \qquad \text{(FOL 27)}$$

Example:

A constraint is set to restrict the simultaneous usage of *assembly_area_1* resource by *assemble_clip_reading_lamp* and *assemble_hand* activities.

```
simultaneous_use_restriction(

assemble_clip_reading_lamp, assemble_hand,
assembly_area_1).
```
(EX 58)

```
simultaneous_use_restriction(bake_anchovie_pizza,
bake_pepporoni_pizza, oven_1).
```
(EX 59)

4.2.16 Resource configuration: resource_configuration(R, C, A).

Definition:

This term specifies the configuration of a resource with respect to an activity. This term implies that the resource must have the specified configuration for the activity. Moreover, after the completion of the activity, **C** is going to be the configuration of the resource unless changed.

Semantics and Implementation:

This term is a ground term with three arguments:

- **R**: resource ID
- **C**: ID specifying the configuration of the resource
- **A**: activity with which the resource has **C** configuration

```
resource_configuration(R, C, A).
```
(PRO 38)

This implies that if activities a_1 and a_2 require a resource and both activities requires different configuration implying that the resource can not be committed to both activities simultaneously (i.e there exist a simultaneous use restriction constraint).

$$(\forall\, a1, a2, r)\ (\exists\, q1, q2, ti1, ti2, c1, c2, \mathit{rate}, \mathit{rate\_2}, \mathit{unit})$$

$$\mathit{(use\_spec(r, a1, ti1, q1, rate, unit) \vee consumption\_spec(r, a1, ti1, q1, rate, unit)) \wedge}$$

$$\mathit{(use\_spec(r, a2, ti2, q2, rate\_2, unit) \vee consumption\_spec(r, a2, ti2, q2, rate\_2, unit)) \wedge}$$

$$\mathit{resource\_configuration(r, c1, a1) \wedge \neg\, resource\_configuration(r, c1, a2) \supset}$$

$$\mathit{simultaneous\_use\_restriction(a_1, a_2, r)} \qquad \text{(FOL 28)}$$

Example:

```
resource_configuration(injection_molder, c1, A).     (EX 60)
```

The above assertion specifies that the *injection molder* resource must have *c1* configuration when associated with activity *A*.

4.2.17 Committed to: committed_to(R, A, S, Ti, Amount_committed, Unit)

Definition:

This predicate specifies the commitment of a resource to an activity thereby making the resource partly/fully unusable/inconsumable by any other activity. A resource is committed to an activity as a result of a scheduling activity.

Semantics and Implementation:

Committed_to is defined as a ground term with the following parameters:

- **R**: ID of the resource to be used/consumed.
- **A**: ID of the activity to use/consume the resource.
- **S**: the ID of the state that is satisfied by the assertion of the committed term.
- **Ti**: the ID of the time interval of the activity.
- **Amount_committed**: amount of the resource that is committed to an activity.
- **Unit**: unit of measurement.

```
committed_to(R, A, S, Ti, Amount_commited, Unit). (PRO 39)
```

where the variable *Amount_Committed* represents:

- the number of resource's objects if the resource is being consumed.
- the number of capacity units being committed if the resource is being used.

"A constraint on the *committed to* term is that the time interval of commitment should either be equal or greater than that is defined in the specification".

$$\mathit{(\forall r, a, s, ti, ti_2, q, q', rate, unit)\ committed\_to(r, a, s, ti, q', unit) \wedge}$$

$$(consumption\_spec(r, a, ti_2, q, rate, unit) \vee use\_spec(r, a, ti_2, q, rate, unit)) \supset (contains(ti, ti_2) \vee equal(ti, ti_2)) \quad \text{(FOL 29)}$$

Example:

If used to specify the commitment of a resources, being consumed by an activity:

```
committed_to(clip_base, fabricate_clip,
consume_clip_base_for_clip, pd1, 1, object).          (EX 61)
```

One *clip_base* resource is committed to be consumed by *fabricate_base* activity at time interval *pd1.*

```
committed_to(clip_base, fabricate_socket_seat,
consume_clip_base_for_socket_sear, pd1, 1, object). (EX 62)
```

One *clip_base* resource is committed to be consumed by *fabricate_socket_seat* activity at time interval *pd1.*

If used to specify the commitment of a resources being used by an activity:

```
committed_to(assembly_area_1,assemble_hand,
use_assembly_area_1_forassemble_hand,pd1,20,block_1).(EX 63)
```

Twenty unit capacity of the *assembly_area_1* resource is committed to be used by the activity *assemble_hand* at time interval *pd1.*

```
committed_to(hammer_1, assemble_hand, use_hammer_1_for_
assemble_hand,pd1, 1, object).                      (EX 64)
committed_to(hammer_1, assemble_base,
use_assembly_area_1_for_assemble_base,pd2,1,object).(EX 65)
committed_to(assembly_area_2, assemble_base,
use_assembly_area_2_for_assemble_base,pd2,20,object).(EX 66)
```

4.2.18 Total Committed: total_committed(R, TQ, Tp, Unit)

Definition:

This predicate specifies the total amount committed of a resource to all activities at a specified time point.

Semantics:

The total commitment of a resource is defined to be the summation of all amount committed of resources to all activities at time point t. The first order logic would be in the form of:

$$(\forall\, r, tp, u)\ (\exists\, TQ)\ total\_committed(r, TQ, tp, u) \equiv (\exists\, pd1, pd2 \ldots pdn, a_1, a_2 \ldots a_n, q_1, q_2 \ldots q_n)$$
$$rknown(r) \wedge$$

$$committed\_to(r, a_1, pd1, q1, u) \wedge period\_contains(ti, pd1) \wedge$$

$$(committed\_to(r, a_2, pd2, q_2, u) \wedge \ldots\ldots \wedge committed\_to(r, a_n, pdn, q_n, u) \wedge$$
$$TQ = q_1 + q_2 + \ldots + q_n \qquad \text{(FOL 30)}$$

"An effect of a resource being committed is that after the completion of the activity the total amount committed will be decremented"[1].

$$(\forall\, s, r, a, q, q', ti, tp, tp', u)$$
$$((use(s, a) \wedge uses(s, r)) \vee (consume(s, a) \wedge consumes(s, r))) \wedge total\_committed(r, q', tp, u) \wedge$$
$$enabling\_state(s, tp, possible) \wedge (tp = SP(ti)) \wedge$$
$$(tp' = EP(ti)) \supset total\_committed(r, q - q', tp', u) \qquad \text{(FOL 31)}$$

Implementation:

The term is defined with four arguments:

- **R**: ID of the resource being checked
- **TQ**: variable through which the total amount committed of the resource, to different activities, is returned.
- **Tp**: ID of the time point of commitment.
- **Unit**: unit of measurement.

```
total_committed(R, TQ, Tp, Unit):-
   unit_of_measurement(R, Unit_id, Unit, A),
   measured_by(R, Unit_id, A),
   committed_to(R, A2, S, Ti, Amount_Com, Unit),
   period_contains(Ti, Tp),
   tmp_var(TT), TQ1 is TT + Amount_Com,
   retract(tmp_var(TT)),assert(tmp_var(TQ1)),
   fail.                                              (PRO 40)
   total_committed(R, TQ, Tp, Unit):- tmp_var(TQ).    (PRO 41)
```

Example:

Total committed axiom calculates the total commitment of a resource by different activities that are simultaneously using/consuming the resource. The predicate could be used to check the total commitment of a resource *clip_base*, being consumed at time point *tp1* with the use of:

[1] In TOVE, this axiom is called an effect axioms which links the resource ontology with the causal theory of activity [Fadel et al 94].

```
?- total_committed(clip_base, TQ, tp1, object).        (EX 67)
```

the above predicate would return the total commitment of the resource through the variable *TQ* with a value of 2. This is done by satisfying the goals of defined axiom in ((PRO 40), (PRO 41)) and by checking the commitment of the resource to different activities (i.e through checking the committed to ground terms (EX 61) and (EX 62)).

The predicate is used to check the total commitment of a resource *assembly_area_1*, being used at time point *tp1* with the use of:

```
?- total_committed(assembly_area_1, TQ, tp1,
block_1).
                                                    (EX 68)
```

the above predicate, would return the total commitment of the resource through the variable *TQ* with a value of 10 signifying that 10 units of capacity of the resource, *block_1*, is committed to other activities. This is done by satisfying the goals of the total committed axiom ((PRO 40), (PRO 41)) and by checking the commitment of the resource to different activities (i.e through checking the committed to ground term).

4.2.19 Set up time constraint: set_up(R, A1, A2, Dur, Unit)

Definition:

Set-up term specifies the duration required to set-up a resource for usage by an activity. The set-up time is defined with regards to pair of activities. The set-up time includes configuration and location dependent time. In the configuration set-up time, the specified duration as a function of the time required to change the resource's configuration state, as result of the last activity supported, to that implied by the activity requiring the resource. As for the location set-up time, it specifies the duration required for the resource to transport or be transported from one location to another[1].

Semantics:

"Set up time is equal to the time required to change the resource's configuration. If the resource needs to be relocated, then the set up time also includes the time of transportation"

$$(\forall\, r, a_2, l_2, dur, u)\ set\_up(r, a_2, l_2, dur, u) \equiv (\exists\, a_1, ti, q, c, tp_1, l_1, ct, lt, u_2, s_1)$$

[1] OPIS [Smith 89] includes the notion of defining a set up constraint which is defined in terms of location and/or configuration dependent set up times.

$$committed\_to(r, a_1, s_1, ti, q, u) \wedge$$
$$tp = EP(ti) \wedge resource\_configuration(r, c, a) \wedge rpl(r, q, tp_1, l_1, u) \wedge$$
$$(config\_set\_up(r, a_1, a_2, ct, u_2) \wedge loc\_set\_up(r, l_1, l_2, lt, u_2) \wedge dur = ct + lt) \quad \text{(FOL 32)}$$

The set-up duration is the summation of the configuration and location dependent set-up times. If the resource is not to be moved from a location[1], then the set up time is only the time needed to re-configure the resource.

Configuration set-up time (config_set_up) is defined as a ground term with these arguments:

- **R**: resource ID
- **A1**: the activity that caused the last configuration change of the resource
- **A2**: activity requiring the change in the configuration of the resource
- **CT**: configuration dependent set-up time
- **Unit**: temporal unit of measurement

```
config_set_up(R, A1, A2, CT, Unit).                    (PRO 42)
```

Location set-up time (loc_set_up(R, A, L1, L2, CL, Unit)) is defined as a ground term with six arguments:

- **R**: resource ID
- **L1**: the location from which the resource is to be relocated
- **L2**: destination of the resource
- **LT**: location dependent set-up time
- **Unit**: temporal unit of measurement

```
loc_set_up(R, L1, L2, LT, Unit).                       (PRO 43)
```

Implementation:

The "set up" term is defined with six arguments:

- **R**: the resource ID
- **A1**: the activity requiring the present configuration that is to be changed
- **A2**: is the activity that requires the resource and requires a new configuration

[1] i.e already in the location required by the activity

- **L2**: the location in which the resource is going to used
- **Dur**: set-up time duration
- **U**: temporal unit of the duration

```
set_up(R, A2, L2, Dur, U):-
    rknown(R),
    find_current_resource_config_and_location(R, A1, L1),
    ((config_set_up(R, A1, A2, CT, U),
    loc_set_up(R, L1, L2, CL, U),
    Dur = CT + CL);
    ((config_set_up(R, A1, A2, CT, U), Dur = CT);
    (loc_set_up(R, L1, L2, CL, U), Dur = CL))).            (PRO 44)
find_current_resource_config_and_location(R, A1, L1):-
    findall(Tp, committed_to(R, A1, S, Ti, Q, U), List),
    find_max(List, Tp_max),
    resource_configuration(R, C, A1),
    findall(Tp, rpl(R, Q, Tp, L1, U), List),
    find_max(List, Tp_max),
    rpl(R, Q, Tp_max, L1, U).                              (PRO 45)
```

Example:

When the following is queried:

```
?- set_up(injection_molder, fabricate_plug_on_wire, _,
Dur, minute).
                                                           (EX 69)
```

the variable ***Dur*** will be instantiated to **60** minutes. This set-up time corresponds to the required time to change the configuration of the resource to suite the configuration required by *fabricate plug on wire* activity.

```
config_set_up(injection_molder, fabricate_wire,
fabricate_plug_on_wire, 60, minute).                       (EX 70)
```

Since the resource is a stationary resource therefore no "location set up" is not asserted for the resource with respect to the activity.

4.2.20 Capacity recognition process:

Capacity is defined to be the maximum set of activities that can simultaneously use/consume a resource at a specific time. In the case where the resource is functionally indivisible then the capacity denotes an activity that could use/consume the resource. On the other hand if the resource is functionally divisible[1], capacity represents the number of activities that a resource can support simultaneously.

The complexity of the process of determining the capacity of a resource depends on the activities requiring the resource and the activities already supported by the resource. The capacity recognition process is solvable in polynomial time in the case where the activities using/consuming the resource are homogenous. Homogeneity implies that activities require equivalent amounts of the resource or processing time or integral multiples thereof[2]. Accordingly the output of the capacity recognition process is reducible to a number which represents the number of activities that the resource could be allocated to. The process becomes complex in the case where the activities requiring the resource are heterogenous. Finding the maximum set of activities is reducible to a single machine scheduling problem which is NP-hard. If the resource is functionally divisible, then the process becomes NP-hard in the strongest sense as the resource can support multiple activities simultaneously. What is required is a sequencing heuristic for activities that use or consume a resource in a predetermined time window. The sequencing heuristic is to be defined for a certain objective such as to lessen the number of tardy activities. The issue of defining a heuristic is dealt with in the capacity recognition chapter. The sequencing heuristics have been defined and implemented using C programming language. The implementation takes an input, a set of activities with their respective starting times, due dates, processing times and resource capacity requirement. The output of the code is the sequence of activities that are to use or consume the resource and hence defining the maximum set of activities that can be simultaneously supported by a resource.

In the ontology, three axioms are defined for usage in the capacity recognition process:

1. has current activity
2. available for
3. available capacity

4.2.20.1 Has current activity: has_current_activity(R, Act_list, Tp)

Definition:

[1] implying having the ability being shared by multiple activities.

[2] in this thesis, homogenous activities are considered as occurring over the same time period and require equivalent resource amounts.

"has current activity" predicate specifies the activity(ies) supported[1] by a resource at the specified time point *Tp*. If the resource is sharable then *has_current_activity* would return a list of activities the are using/consuming the resource.

Semantics:

"An activity is supported by a resource if the resource is committed to the activity for a time interval that includes the time point of the check and if the enabling state of the activity has enabled status"

$$(\forall r)\ (\exists\ act\_list,\ tp) has\_current\_activity(r,\ act\_list,\ tp) \equiv$$

$$\exists(ti,\ s,\ q,\ u,\ a)\ (committed\_to(r,\ a,\ s,\ ti,\ q,\ u) \wedge (period\_contains(ti,\ tp)$$

$$\wedge\ enabling\_state(s,\ tp,\ enabled). \qquad \text{(FOL 33)}$$

Implementation:

This predicate is defined with three arguments:

- **R**: ID of the resource being checked.
- **Act_List**: List of activities using or consuming the resource
- **Tp**: ID of the specified time point at which the resource is being checked.

```
has_current_activity(R, Act_List, Tp):-

    setof(A, check_commitment(R, A, Tp), Act_List).      (PRO 46)

check_commitment(R, A, Tp):-

    committed_to(R, A, S, Ti, Q, U),

    period_contains(Ti, Tp), enabling_state(S, Tp,
    enabled).
                                                          (PRO 47)
```

Example:

```
    has_current_activity(assembly_area_2, Act_List, tp2).(EX 71)
```

When the resource *assembly_area_2* is checked whether it supports any activities[2] or not at time point *tp2*, the variable **Act_list** will be returned with a value, **assemble_base**, as this activity uses the resource at the specified time point.

```
    has_current_activity(assembly_area_2, Act_List, tp4).(EX 72)
```

[1] i.e activity to which the activity is committed to

[2] i.e activities to which the resource is committed to

When the resource *assembly_area_2* is checked whether it supports any activities or not at time point *tp4*. ***No*** will be returned as no *committed to* predicate is asserted that indicates that no activity is using/consuming the resource at time point *tp4*.

4.2.20.2 Availability for a set of activities: available_for(R, [A], T, Capacity)

Definition:

The *availability for* predicate checks whether a resource could support a set of activities or not. The output of the predicate is the maximum set of activities that could be supported by a resource.

Semantics:

"A resource is available to a set of ordered activities if the resource has the capacity to support them and there is no simultaneity constraint among the set of activities requiring the resource and the ones already supported by the resource"

For a fixed set of activities **[a]**, *available for* is defined in the form of:

$$(\forall r)\ (\exists a, ti,)\ available\_for(r, [a], ti) \equiv$$

$$(\forall\ tp \in ti)\ (\exists\ unit\_id, u, amt\_required, tq, q, amount_1, amount_2 \ldots amount_n, rate_1, rate_2 \ldots rate_n)$$

$$(consumption\_spec(r, a_1, ti, amount_1, rate_1, unit) \vee use\_spec(r, a_1, ti, amount_1, rate_1, unit)) \wedge$$

$$(consumption\_spec(r, a_2, ti, amount_2, rate_2, unit) \vee use\_spec(r, a_2, ti, amount_2, rate_2, unit)) \wedge \ldots\ldots$$
$$\wedge$$

$$(consumption\_spec(r, a_n, ti, amount_n, rate_n, unit) \vee use\_spec(r, a, ti, amount_n, rate_n, unit)) \wedge$$

$$amt\_required = amount_1 + amount_2 + \ldots\ldots + amount_n \wedge$$

$$(period\_contains(ti, tp) \wedge (total\_committed(r, tq, tp, unit)) \wedge unit\_of\_measurement(r, unit\_id, u, a) \wedge$$

$$measured\_by(r, unit\_id, a)\ \wedge rp(r, q, tp, unit)\ \wedge (amt\_required \geq q- tq) \wedge$$
$$((\forall a_x \in a)\ no\_restricition(r, a, a_x, ti)) \qquad \text{(FOL 34)}$$

$$no\_restricition(r, a, a_x, ti) \equiv (committed\_to(r, a_x, s, ti_2, q, u) \wedge$$
$$(a \neq a_x) \wedge period\_overlaps(ti, ti_2) \wedge \neg(simultaneous\_use\_restriction(a, a_x, r)))\text{(FOL 35)}$$

The above definition is based on the underlying assumption that the geometry of the resource does not have a role in defining its capacity. This is also the assumption on which the sequencing heuristic is based on, as defined in chapter five.

Implementation:

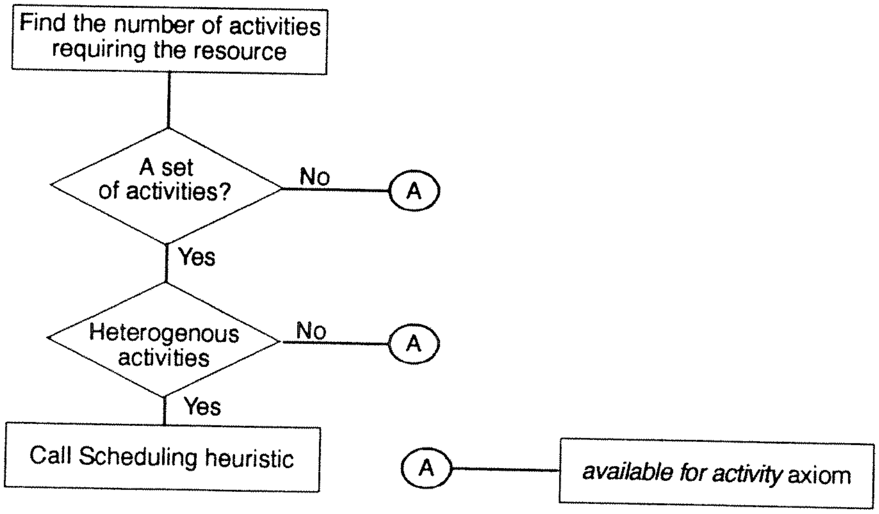
The implementation of the availability term is different from the first order logic formulation as the implementation returns the maximum set of activities that can be supported by a resource. On the other hand, the FOL formulation specifies only whether a resource can support an ordered set of activities or not.

If a resource is required by a set of **homogenous activities**, then *available for homogenous* axiom is called. This is because in the homogenous case, the capacity recognition process is reducible to a number denoting the maximum number of activities that could be supported by a resource. However, in the case when having **heterogenous activities**, then the capacity recognition process is reducible to a single machine scheduling problem. The scheduling heuristic is implemented in C programming environment and called from Prolog.

The respective pseudo-code for *available for* axiom is:

```
Find the number of activities requiring the resource
If the number = 1, invoke available for activity axiom
Else
If (the activities requiring the resource and the activities already supported by the resource homogenous) & (no simultaneous use restriction exists)
Then invoke available for homogenous axiom
Else invoke one machine scheduling heuristic.
```

FIGURE 43 Capacity recognition process


Find the number of activities requiring the resource
A set of activities?
No
A
Yes
Heterogenous activities
No
A
Yes
Call Scheduling heuristic
A
available for activity axiom

The requirement/need of an activity is checked using either the *consumption_spec* or the *use_spec* predicates. The available capacity is calculated through deducting the total 'amount' committed of the resource from the resource point of the resource over the whole specified time interval *Ti*. Moreover the *simultaneous_use_constraint* is used to check if there exist a constraint on two activities to use/consume a resource simultaneously.

The term is defined with the following arguments:

- **R**: resource that is required to be used or consumed
- **A**: the activity or activities requiring the resource
- **Ti**: resource time window
- **Capacity**: the variable the returns that activities that could be supported by the resource.

The implementation of the above pseudo code in Prolog is as follows:

```
available_for(R, A, Ti, Capacity):-
   rknown(R),
   length(A, Len), % Calculate the length of the A's list
   (
   ((Len == 1), available_for_activity(R, A, Ti));
   ((Len \= 1), (
    homogenous_or_heterogenous(R,A,Ti,Well),
   % homogenous/heterogenous case?
   (
   (Well == homogenous,
   time_period(Ti, ST, ET, MinDUr, Dur, MaxDur),
   time_point(ST, StMin, StMax),
   time_point(ET, EtMin, EtMax),
   Counter is StMin, Stop is EtMax-1,
   available_for_homogenous(R, A, Ti, Len, Counter, Stop,
   Needed_Capacity, Tmp3),
   printf(A,1, Tmp3, Result)% print all supported activities
   );
   (Well == heterogenous,
   Printf('Activities_are_not_homogenous'),
    Printf('Use_sequencing_heuristic'),
    ))))).                                                        (PRO 48)

  homogenous_or_heterogenous(R,A,Ti,Well):-
    time_period(Ti, ST,_,_,_,_),
    % if the resource isn't supporting any activities then
    % check homogeneity among the A list
    ((\+ has_current_activity(R,_,ST,Act_list),
    (compare_list(R,A,Ti,Well);
    (\+ compare_list(R, A, Ti, Well),
    Well = heterogenous)),!);
    (has_current_activity(R,_,ST,Act_list),
    (compare(R,A,Act_list,Ti) -> Well = homogenous;
    Well = heterogenous)
    )).                                                           (PRO 49)

compare_list(R, [], Ti, Well). % Stopping criterion

compare_list(R, [H2|T2], Ti, Well):-
    compare_tail(R, H2, T2, Ti, Well),Well =
    homogenous.                                                   (PRO 50)

compare_tail(R, H2, [], Ti, Well).

compare_tail(R, H2, [H3|T3], Ti, Well):-
    (use_spec(R,H2,Ti,Amt_Required2,Rate,U);
    consumption_spec(R,H2,Ti,Amt_Required2,Rate,U)),
    (use_spec(R,H3,Ti,Amt_Required3,Rate,U);
    consumption_spec(R,H3,Ti,Amt_Required3,Rate,U)),
    Amt_Required2 == Amt_Required3,
    no_restricition(R, H2, H3, Ti),
    compare_tail(R, H2, T3, Ti, Well).                            (PRO 51)


compare(R,[],Act_list,Ti). %Stopping criterion

compare(R,[H|T],Act_list,Ti):-
```

```
    over_all_supported_As(R,H,Act_list,Ti),
    compare(R,T,Act_list,Ti).                               (PRO 52)


over_all_supported_As(R,H,[],Ti). %Stopping criterion

 over_all_supported_As(R,H,[Head2|Tail2],Ti):-
    committed_to(R,Head2,Ti2,Amt,_), overlaps(Ti,Ti2),
    (use_spec(R,H,Ti,Amt_Required,_,U) |
    consumption_spec(R,H,Ti,Amt_Required,_,U)),
    Amt == Amt_Required,
    over_all_supported_As(R,H,Tail2,Ti).                    (PRO 53)


available_for_homo(R,[H|T],Ti, Len, Counter, Stop, Needed_Capacity, Tmp3):-
    time_point(Tp,Counter,_),
    (use_spec(R,H,Ti,Amt_Required,_,U);
    consumption_spec(R,H,Ti,Amt_Required,_,U)),
    ((var(Tmp3), Needed_Capacity is Len*Amt_Required);
    Needed_Capacity is (Len-Tmp3)*Amt_Required),
    clear_rp, rp(R,Q,Tp,U),
    clear_com, ((\+ total_Com(R,TQ,Tp1,U) -> TQ is 0);
    (clear_com,total_Com(R,TQ,Tp,U))),
    Tmp1 is Q - Needed_Capacity - TQ,
    (Tmp1 >= 0 -> Result = [H|T];
    (Available is Q-TQ, Afford is Available div Amt_Required,
    Not_supported is Len-Afford,
    retractall(dum(_)), assert(dum(Not_supported)))),
    Dummy is Counter +1,
    ((var(Not_supported), \+ var(Tmp3),Tmp is Tmp3);
    var(Not_supported), var(Tmp3), Tmp is Len);
    (\+ var(Tmp3), \+ var(Not_supported), dum(X), Tmp is X,
    available_for_homo(R, [H|T], Ti,Len, Dummy, Stop, Y,
    Tmp));
    (dum(X), Tmp3 is X)),
    ((Counter == Stop -> !) ;
    available_for_homo(R, [H|T], Ti,Len, Dummy, Stop, Y,
    Tmp3)).                                              (PRO 54)


printf(A,ST,Tmp3,Result):-ST>Tmp3,Result=A. %Stopping Crite-
rion

printf(A,ST,Tmp3,Result):-
    Dum is 1+ST,del_item(A,FF),
    printf(FF,Dum,Tmp3,Result).                          (PRO 55)


del_item([Head|Rest],Rest).

del(List, ST, Tmp3, Rest):-
    del_item(List,Rest),
    ((ST == Tmp3 -> !); Dum is 1+ST, del(List,Dum,Tmp3,Rest),
    AA = [H|T],AA = Rest).                               (PRO 56)
```

Example:

"available for" axiom could be used to check the availability of a resource:

- to an activity
- to a set of activities
- to a set of activities with different application types. That is to say the resource would be partly consumed by a number of activities.

If one activity requires a resource:

When the predicate is used to check the availability of the resource *clip_base* for activity *assemble_clip_base* over time interval *pd1*:

```
?- available_for(clip_base, [assemble_clip_base], pd1,
Capacity).                                              (EX 73)
```

the above will return the variable **Capacity** instantiated to **[assemble_clip_base]** indicating that the resource *clip_base* is available for activity *assemble_clip_base* over the time period *pd1*. This is done through satisfying the goals of axioms (PRO 48) by checking:

- resource point of the resource at the specified time, (EX 48), specifying that it is equal to four units.
- consumption specification of the activity, (EX 27), which specifies that the activity needs one unit of the resource,
- the total amount committed of the resource, through using *total_committed* axiom. In this case the total commitment of the resource is equal to two units of the resource as two units are committed to activities *fabricate_clip* and *fabricate_socket_seat.*

When the predicate is used to check the availability of the resource assembly_area_1 *for activity assemble_clip_base at time interval pd1*:

```
?- available_for(assembly_area_1, [assemble_clip_base_2],
pd1, Capacity).                                    (EX 74)
```

the above will return *No* indicating that the resource *assembly_area_1* is not available for activity *assemble_clip_base* over the time period *pd1* because of the lack of sufficient units of capacity over the whole time period *dp1*. This is done through satisfying the goals of *available_for* axiom by checking:

- the resource point of the resource specifying that the amount is one hundred,
- use specification of the activity, (EX 28), which specifies that the activity requires ninety unit capacity.
- the total amount committed of the resource. The total amount committed is equal to twenty unit capacity, as the resource is committed to activity *assemble_hand.*

If there exist a simultaneous use constraint for two activities requiring a resource:

```
?- available_for(assembly_area_1, [assemble_hand], pd1,
Capacity).                                         (EX 75)
```

If the above predicate is used to check of resource *assembly_area_1* is available for activity *assemble_hand* at time interval *pd1*, the returned value will be *No* as activity *assemble_clip_reading_lamp* uses the activity at the same time interval and there is a simultaneous use restriction of the two activities .

If multiple activities require a resource simultaneously:

'available for' could also check the availability of a resource for multiple activities characterized by having different application types: use and consume. Consider having two activities:

- *heat up water*, using a solar water heater (SWH), that uses water stored in the SWH tank.
- flush_water that consumes the water in the SWH tank.

The activities require the same resource (water) at the same time but both activities are constrained to operate with minimum amount of water that should be available in the tank. Assuming that no water is being added to the tank during the execution of the two activities, then the support of the resource (water) to any of the activities depends on having the capacity to fulfill the requirements.

```
?- available_for(water, [heat_up, flush_water], pd3,
Capacity).                                              (EX 76)
```

Both activities are homogenous because they require same amount of the resource.

```
use_spec(water, heat_up, pd3, 3, 1, rate_1)[1]          (EX 77)
consumption_spec(water, flush, pd3, 3, 1, rate_1)       (EX 78)
```

available for would return **Capacity = [heat_up, flush_water]** specifying that the resource can support both activities simultaneously.

In the case of heterogenous activities, the user will be prompted to use the sequencing heuristic defined in the capacity chapter.

4.2.20.3 Available Capacity: available_capacity(R, Tp, Amount_Available, Unit)

Definition:

The predicate *available capacity* is defined as the amount of a resource that physically exists and it is not committed to any other activity at a specific time point. Furthermore in the case where a resource is *divisible and continuous* then *available capacity* denotes to the portion of the resource that could be used/consumed. On the other hand in the case where a resource is *indivisible* then the available capacity denotes if the resource is occupied or not.

This predicate, *available capacity*, specifies the available amount of a resource by calculating the uncommitted amount at the specified time point.

[1] object(rate_1, 1, L, minute)

Semantics:

"To check availability, the resource has to be known with a resource point defined at a specified time point. The available amount is calculated from the difference of the resource point and the amount committed of the resource at the specified time point".

$$(\forall\, r)\ (\exists\, r, tp\ amount, u)\ available\_capacity(r, tp, amount, u) \equiv (\forall\, a)(\exists\, tq, q, unit\_id)\ rknown(r) \wedge$$
$$rp(r, q, tp\ u) \wedge$$
$$total\_committed(r, tq, tp, u) \wedge unit\_of\_measurement(r, unit\_id, u, a) \wedge$$
$$measured\_by(r, unit\_id, a) \wedge amount \geq q - tq \qquad \text{(FOL 36)}$$

Implementation:

```
available_capacity(R, Tp, Amount_Available, Unit):-
    (rknown(R),
    rp(R, Q, Tp, Unit),
    total_committed(R, TQ, Tp, Unit),
    Amount_Available is Q - TQ,
    assert(ad(Amount_Available)), fail.                    (PRO 57)
available_capacity(R,Ti,Amount_Available, Unit):-
    ad(Amount_Available).                                  (PRO 58)
```

Example:

When the predicate is used to check the availability of the resource *clip_base* at time point *tp1*:

```
?- available_capacity(clip_base, tp1, Amount_Available,
Unit).
```
(EX 79)

will return the value of 2 which represents the available amount (not committed) of the resource *clip_base* at time point *12* through satisfying the goals of *available capacity* axiom. The goals are satisfied using the assertion of the resource point of *clip base*, declared in (EX 48), and the *committed to* assertions declared in (EX 61) and (EX 62).

When the predicate is used to check the availability of the resource *assembly_area_1* at time point *tp1*:

```
?- available_capacity(assembly_area_1, tp1,
Amount_Available, Unit).
```
(EX 80)

will return the value of *80* which indicates that eighty units of the resource *assembly_area_1* is available (not committed) at time point *tp1* through satisfying the goals of *available capacity axiom.*

4.2.21 Trend: trend(R, Tp, Result)

Definition:

Trend predicate indicates whether the capacity of a resource is decreasing, increasing or at a steady state[1].

Semantics:

The capacity trend of a resource, at a specific time point, is determined by calculating the rate of change of the resource point over a time interval preceding the time point of check.

$$\frac{d}{dt}\, rp(R,\ Q,\ Tp,\ Unit)$$

The capacity trend of a resource could be either:

- decreasing,
- increasing,
- or steady

The capacity of a resource could be *decreasing*:

- if the rate of change of the resource point is decreasing over a time period before the specified time point.

$$(\forall\ r)\ (\exists\ tp)\ trend(r,\ tp,\ decreasing) \equiv$$
$$(\forall a\ \exists rate)\ (rp\_at\_last\_tps(r,\ a,\ tp,\ rate) \wedge (rate < 0.00)) \qquad \text{(FOL 37)}$$

The capacity of a resource could be *increasing*:

[1] Qualitative physics research presents means to make programs that interact with the world as well as people do. Efforts such as [Forbus 84] and [Kuipers 84] strive to define a methodology for describing complex systems. Forbus defines an ontology for the transfer of causality. Kuipers' research, on the other hand, is concerned with "qualitative simulation of physical systems whose descriptions are stated in terms of continuously varying parameters".

- if the rate of change of the resource point is increasing over a time period before the specified time point.

$$(\forall\ r)\ (\exists\ tp)\ \textit{trend}(r,\ tp,\ \textit{increasing}) \equiv$$

$$(\forall a\ \exists \textit{rate})\ (\textit{rp\_at\_last\_tps}(r,\ a,\ tp,\ \textit{rate}) \wedge (\textit{rate} > 0.00)) \qquad \text{(FOL 38)}$$

The capacity of a resource could be *steady*:

- if the rate of change of the resource point is steady over a time period before the specified time point[1].
- the resource was not committed in the previous time points

$$(\forall\ r)\ (\exists\ tp)\ \textit{trend}(r,\ tp,\ \textit{steady}) \equiv$$

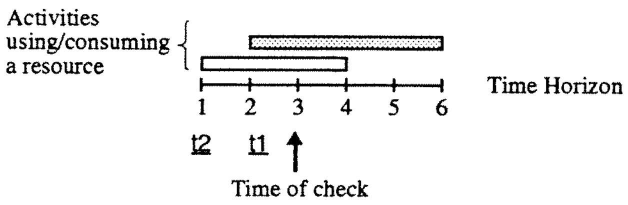
$$(\forall A)\ (\exists\ \textit{amount},\ \textit{unit})\ (\textit{rp\_at\_last\_tps}(r,\ a,\ tp,\ \textit{rate}) \wedge (\textit{rate} = 0.00)) \vee$$

$$\neg(\textit{committed\_to}(r,\ a,\ s,\ ti,\ \textit{amount},\ \textit{unit}) \wedge \textit{period\_contains}(ti,\ tp)) \qquad \text{(FOL 39)}$$

$$\textit{rp\_at\_last\_tps}(r,\ a,\ tp,\ \textit{rate}) \equiv (\exists\ q_1,\ q_2,\ tp_1,\ tp_2)$$

$$\textit{rp}(r,\ q_2,\ tp_2,\ \textit{unit\_id}) \wedge \textit{rp}(r,\ q_1,\ tp_1,\ \textit{unit\_id}) \wedge (\textit{rate} = (q_1 - q_2)/(tp_1 - tp_2)) \qquad \text{(FOL 40)}$$

FIGURE 44 Trend - time horizon



Implementation:

The respective pseudo code for the trend axiom is:

```
Find the resource point of the resource ($rp_1$) at first time point ($t_1$)
Find the resource point of the resource ($rp_2$) at second time point ($t_2$)
Find the rate of usage drp/dt = ($rp_1$ -$rp_2$)/($t_1$ - $t_2$)
```

[1] The choice of the two time point (tp_1 & tp_2) are arbitrary and the issue of selecting these time points depends on the granularity the user requires.

Case:

(Rate > 0.00) **Then** the *TREND* is increasing

(Rate < 0.00) **Then** the *TREND* is decreasing

(Rate = 0.00) **If** the resource was not committed over the ti interval bounded by the two time points **Then** the *TREND* is steady

The implementation of the above pseudo-code in Prolog is as follows:

```
trend(R, Tp, Result):-
    rknown(R),
    time_point(Tp,StMin,StMax),
    time_period(Ti, ST, ET, MinDur, Dur, MaxDur),
    StMin1 is StMin -1, StMin2 is StMin -2,
    time_point(Tp1, StMin1, _),
    ((StMin1 == 0 -> Result = undetermined,!);
    (\+ committed_to(R, A, S, Ti, Q, U),
    period_contains(Ti, StMin1),
    \+ committed_to(R, A, S, Ti, Q, U), period_contains(Ti,
    StMin2), Result = steady,!);
    rp(R, Q1, Tp1, Unit),
    retractall(quan(_)), assert(quan(Q1)),
    time_point(Tp2, StMin2, _),
    rp(R, Q2, Tp2, Unit),
    assert(quan(Q2)),
    Drp is Q1 - Q2, Dt is StMin1 - StMin2,
    Rate is Drp/Dt,
    (((Rate > 0.000), Result = increasing),!|
    ((Rate < 0.000), Result = decreasing),!;
    ((Rate == 0.000), Result = steady,!))).                (PRO 59)
```

Example:

So, two predicates could be used for finding out the trend of the capacity if the resource. One predicate (*strong_trend*) gives a stronger indication, of the trend, than the *trend* predicate. If the *trend* predicate is used and indicated that a resource has a steady trend, there is no guarantee that the use of the *strong_trend* predicate would give the same an indication.

```
rp(oven, bake_large_pizza, tp2, 60, block_1).          (EX 81)
rp(oven, bake_large_pizza, tp1, 60, block_1).          (EX 82)
rp(oven, bake_medium_pizza, tp7, 40, block_1).         (EX 83)
rp(oven, bake_medium_pizza, tp8, 60, block_1).         (EX 84)
rp(pepperoni, 40, tp8, object).                        (EX 85)
rp(pepperoni, 60, tp7, object).                        (EX 86)
```

```
?- trend(oven, tp3, Result).                           (EX 87)
```

The use of the above will return the variable **Result** with an instantiated value which is **steady**. This is done through satisfying the goals of trend axiom (PRO 59) with use of the *rp* predicate specified in (EX 81) and (EX 82).

```
?- trend(oven, tp9, Result).                           (EX 88)
```

The use of the above will return the variable **Result** with an instantiated value which is **increasing**. This is done through satisfying the goals of trend axiom (PRO 59) with use of the *rp* predicate specified in (EX 83) and (EX 84).

```
?- trend(pepperoni, tp9, Result).                      (EX 89)
```

The use of the above will return the variable **Result** with an instantiated value which is **decreasing**. This is done through satisfying the goals of trend axiom (PRO 59) with use of the *resource point* predicate specified in (EX 85) and (EX 86). The *resource point* predicates specify that the resource *pepperoni* has forty and sixty units at the last previous time points which makes the resource to have a decreasing trend.

4.2.22 Activity history: activity_history(R, Act_List, Tp)

Definition:

This predicate specifies the history of usage or consumption of a resource before a specified time point. A list of activities that were supported by the resource will be returned.

Semantics:

"An activity will be included in the list of activities, that were supported by the resource, if the resource was committed to the activity for a time period with end time less or equal than the specified time point".

$$(\forall r)(\exists act\_list, tp)\ activity\_history(r, act\_list, tp) \equiv (\forall a \in act\_list)(\exists ti, q, u, a, s)$$
$$(committed\_to(r, a, s, ti, q, u) \land period\_before(ti, tp) \land enabling\_state(s, tp, completed) \qquad \text{(FOL 41)}$$

Implementation:

This predicate is defined with three predicates:

- **R**: ID of the resource being checked.
- **Act_List**: List of activities that used or consumed the resource
- **Tp**: ID of the specified time point at which the resource is being checked.

```
activity_history(R, Act_List, Tp):-

   setof(A, check_past_commitment(R, A, Tp), Act_List). (PRO 60)


check_commitment(R, A, Tp):-

   committed_to(R, A, S, Ti, Q, U), period_before(Ti, Tp),

   enabling_state(S, tp, completed).                     (PRO 61)
```

Example:

```
?- activity_history(assembly_area_1, Act_List, tp2).(EX 90)
```

When the activity history of resource *assembly_area_1* is queried at time point *tp2*, the variable **Act_list** will be returned with a value equal to **[assemble_clip_reading_lamp, assemble_hand]**, specifying the activities that used the resource prior to time point *tp2*.

4.2.23 Alternative resource: alternative_resource(R, A, List)

Definition:

This term specifies an alternative resource(s) to be used or consumed by an activity. This is useful in the case when an alternative resource is required because of a machine breakdown or unavailability of a resource.

Semantics:

"A resource **(R2)** is an alternative resource for an activity, if the resource is related to a disjunct state that is related to the activity"

$$(\forall r, a)(\exists list)\ alternative\_resource(r, a, list) \equiv (\exists s, s_2, disjunct\_state)\ uses(s_2, r) \land is\_related(s_2, s) \land$$

subclass_of(s, disjunct_state) (FOL 42)

Implementation:

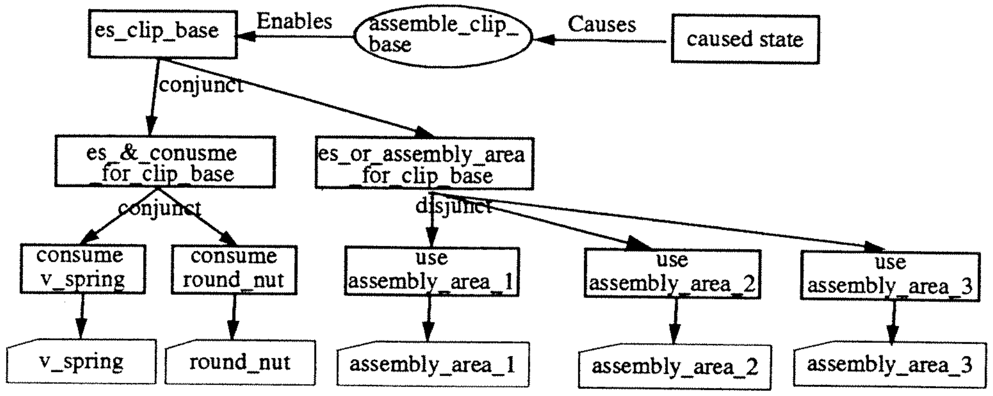
The term is defined with three arguments:

- **R**: resource to which a replacement is required
- **A**: the activity that requires a resource replacement
- **List**: list of alternatives

```
alternative_resource(R,A2,List):-
    uses(S2,R), is_related(S2,S),
    subclass_of(S,disjunct_state), disjuncts(S,List1),
    all_states(R,S2,List1,List2),
    get_res(List2,Result),!.                                (PRO 62)
all_states(R,S2,Tmp_list,List2):-
    uses(S2,R), member_of(S2,Tmp_list),
    del_rep(S2,Tmp_list,List2).                             (PRO 63)
del_rep(X, [X|Tail], Tail). % Stop if X is the Head of List
del_rep(X, [Y|Tail], [Y|Tail1]):-
    del_rep(X, Tail, Tail1).                                (PRO 64)
get_res([],Result):- % Stop if List is empty
    write('Result = '),print(Result), ttynl.               (PRO 65)
get_res([X|Tail], List):-
    uses(X,R), add(R,List,Result),get_res(Tail,Result).(PRO 66)
add(R,List, [R|List]).
```

Example:

FIGURE 45 assemble clip base



Reference to figure 45, "assemble clip base" activity can either use "assembly area 1" or "assembly area 2" or "assembly area 3". If an alternative resource for "assembly area" is required through using:

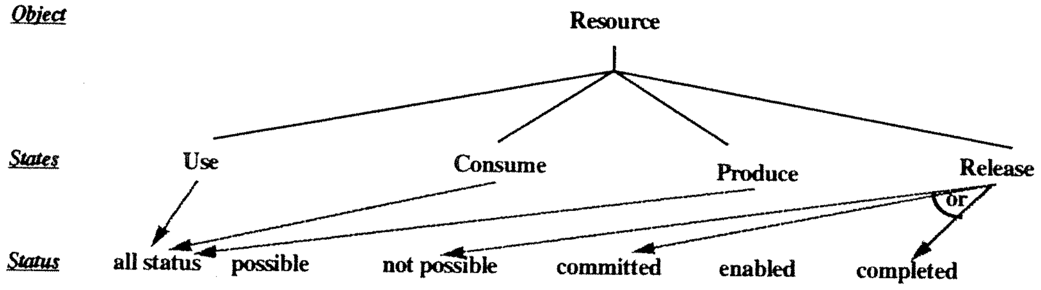
```
alternative_resource(assembly_area_1, assemble_clip_base,
List).
```
(EX 91)

the variable **List** will be bounded to **[assembly_area_2, assembly_area_3]** specifying two alternative resources.

4.3 Relation of the resource ontology with that of the activity-state

This sections presents axioms that define how the activity-state ontology is linked to the resource ontology. Most state's status are defined in terms of resource properties (i.e ontology). These status' definitions are presented in the coming section. [see figure 46]

FIGURE 46 States properties



4.3.1 Enabling states: enabling_state(State_ID, Tp, Status)

Definition:

An enabling state specifies what has to be true in order for an activity to be performed. For example in figure 33, there are two terminal enabling states. A consume state specifies that a resource will not available (at least in the same form or properties) after the completion of an activity. The use state on the other hand specifies that resource will available in the form and properties after the completion of an activity.

The use of the predicate would return the status of the use/states state which in turn is dependent on the status of the resource. Recall that there are five status that a state could have:

- **enabled**: when the resource is being used or consumed[1].
- **committed**: when the resource is committed to be used or consumed.
- **completed**: when the resource is no longer being used or consumed by an activity.
- **possible**: when the resource may be used or consumed as it is available for the activity.
- **not_possible**: the resource can not be used or consumed because of the unavailability of the resource to the activity.

Semantics:

The enabling state is <u>**completed**</u> if:

- the activity is in the activity history of the resource.

$$(\forall\ state\_id)\ (\exists\ tp)\ completed(state\_id,\ tp) \equiv (\exists\ r,\ a,\ act\_list)$$

$$((consume(state\_id,\ a) \wedge consumes(state\_id,\ r)) \vee$$

$$(use(state\_id,\ a) \wedge uses(state\_id,\ r))) \wedge$$

$$activity\_history(r,\ act\_list,\ tp) \wedge member\_of(a,\ act\_list) \qquad \text{(FOL 43)}$$

The enabling state is <u>**enabled**</u> if:

- the activity is in the has current activity list of the resource.

$$(\forall\ state\_id)\ (\exists\ tp)\ enabled(state\_id,\ tp) \equiv (\exists\ r,\ a,\ act\_list)$$

$$((consume(state\_id,\ a) \wedge consumes(state\_id,\ r)) \vee$$

[1] not included in this section since its definition is not dependent the resource ontology

$$(use(state\_id, a) \wedge uses(state\_id, r))) \wedge$$

$$has\_current\_activity(r, act\_list, tp) \wedge member\_of(a, act\_list) \tag{FOL 44}$$

The enabling state is **possible** if:

- the resource is available for the activity and
- the resource has not been committed yet to the activity and
- the activity is not executing.

$$(\forall\, state\_id)\ (\exists\, tp)\ possible(state\_id, tp) \equiv (\exists\, r, a, ti, unit)\ (consume(state\_id, a)\ \vee consumes(state\_id, r)) \vee (use(state\_id, a)\ \vee uses(state\_id, r)) \wedge$$

$$available\_for(r, a, ti) \wedge \neg committed\_to(r, a, state\_id, ti, amount, unit) \wedge period\_contains(ti, tp) \wedge$$

$$\neg activity(a, executing, tp) \tag{FOL 45}$$

The enabling state is **not possible** if:

- the resource is not available for the activity and
- the resource has not been committed yet to the activity and
- the activity is not executing.

$$(\forall\, state\_id)\ (\exists\, tp)\ not\_possible(state\_id, tp) \equiv (\exists\, r, a, ti)$$

$$((consume(state\_id, a) \wedge consumes(state\_id, r)) \vee (use(state\_id, a) \wedge uses(state\_id, r))) \wedge$$

$$\neg available\_for(r, a, ti) \wedge \neg committed\_to(r, a, state\_id, ti, amount, unit) \wedge period\_contains(ti, tp) \wedge$$

$$\neg\, activity(a, executing, tp) \tag{FOL 46}$$

The enabling state is **committed** if:

- the resource is committed to the resource and
- the activity is not executing[1] and
- the activity is not completed[2].

$$(\forall\, state\_id)\ (\exists\, tp)\ committed(state\_id, tp) \equiv (\exists\, r, a, ti)$$

$$((consume(state\_id, a) \wedge consumes(state\_id, r)) \vee (use(state\_id, a) \wedge uses(state\_id, r))) \wedge$$

$$committed\_to(r, a, s, ti, amount, unit) \wedge period\_contains(ti, tp) \wedge$$

$$has\_current\_activity(r, act\_list, tp) \wedge \neg member\_of(a, act\_list) \wedge$$

[1] i.e not included in the "has current activity"

[2] i.e not included in the "activity history"

$$activity\_history(r, list, tp) \wedge \neg member\_of(a, list) \qquad \text{(FOL 47)}$$

Implementation:

The term is defined having three arguments:

- **State_ID**: specifies the ID of the state
- **Tp**: specifies time point of the check
- **Status**: is the argument through which the status is returned.

```
enabling_state(State_ID, Tp, Status):-
   ((use(State_ID, A), uses(State_ID, R));
   consume(State_ID, A); consumes(State_ID, R)),
   (completed(State_ID, R, A, Tp, Re_status),!);
   (enabled(State_ID, R, A, Tp, Re_status),!);
   (possible(State_ID, R, A, Tp, Re_status),!);
   (not_possible(State_ID, R, A, Tp, Re_status),!);
   (committed(State_ID, R, A, Tp, Re_status),!);
   (suspended(State_ID, R, A, Tp, Re_status),!).        (PRO 67)
completed(State_ID, R, A, Tp, Status):-
   activity_history(R, Act_List, Tp),
   member_of(A, Act_List),
   Status = completed.                                  (PRO 68)
enabled(State_ID, R, A, Tp, Re_status):-
   ((consume(State_id, A), consumes(State_id, R)),
   (use(State_id, A), uses(State_id, R))),
   activity_history(R, Act_list, Tp),
   member_of(A, Act_list).                              (PRO 69)
possible(State_ID, R, A, Tp, Re_status):-
   available_for(R, A, Ti),
   \+committed_to(R, A, S, Ti, Amount, Unit),
   activity(A, Tp, Status),
   Status \= executing, period_contains(Ti, Tp).        (PRO 70)
not_possible(State_ID, R, A, Tp, Re_status):-
```

```
        \+available_for(R, A, Ti),
        \+committed_to(R, A, S, Ti, Amount, Unit),
        activity(A, Tp, Status),
        Status \= executing, period_contains(Ti, Tp).        (PRO 71)
    committed(State_ID, R, A, Tp, Re_status):-
        committed(R, A, S, Ti, Amount, Unit),
        period_contains(Ti, Tp),
        has_current_activity(R, Act_list, Tp),
        \+member_of(A, Act_list),
        activity_history(R, List, Tp), \+member_of(A, List).(PRO 72)
```

Example:

```
        ?- enabling_state(assembly_area_1, assemble_hand, tp3,
        Status).                                               (EX 92)
```

The use of the above query returns the **status** of the state as being **completed** as the activity "assemble hand" has already been performed (i.e included in the "activity history" of the resource[1].

```
        ?- use(assembly_area_1, assemble_hand, tp2, Status).(EX 93)
```

Now if the same state is checked at tp2, the **status** is returned as being **enabled** as the activity would be currently being performed.

4.3.2 release state: release_state(State_ID, Tp, Status).

Definition:

In TOVE a release state specifies that a resource which has been designated for usage, by an activity, is available. The release axiom would return the status of the release state. Recall there are three possible status:

- **completed**: when the resource has already been released.
- **committed**: when the resource is committed to be released.
- **not_possible**: when the act of releasing the resource is not possible at the specified time point.

Semantics:

The release state is **completed** if:

[1] i.e the activity was committed in a period that ended before tp3.

- the activity that causes the release state is included in the activity history of the resource.

$$(\forall r, a)\ (\exists tp)\ release\_completed(r, a, tp) \equiv (\exists act\_list)\ (activity\_history(r, act\_list, tp) \wedge member\_of(a, act\_list)) \tag{FOL 48}$$

The release state is **committed** if:

- the activity causing the release state is in the resource current activity list[1].

$$(\forall r, a)\ (\exists tp)\ release\_committed(r, a, tp) \equiv (\forall act\_list)\ has\_current\_activity(r, act\_list, tp) \wedge member\_of(a, act\_List) \tag{FOL 49}$$

The release state is **not possible** if:

- if the resource is not committed to the activity the releases it **OR**
- if the resource is not currently supporting the activity.

$$(\forall r, a)\ (\exists tp)\ release\_not\_possible(r, a, tp) \equiv (\exists state, act\_list, ti, q, u)$$

$$((period\_before(ti, tp) \vee period\_contains(ti, tp)) \supset \neg\ committed\_to(r, a, state, ti, q, u) \vee (has\_current\_activity(r, tp, act\_list) \wedge \neg\ member(a, act\_list)) \tag{FOL 50}$$

Implementation:

The term is defined with three arguments:

- **State_ID**: specifies the ID of the state
- **Tp**: specifies time point of the check
- **Status**: is the argument through which the status is returned.

```
release_state(State_ID, Tp, Status):-
    ((release(State_ID, A); releases(State_ID, R)),
    (release_completed(R, A, Tp, Status),!);
    (release_committed(R, A, Tp, Status),!);
    (release_possible(R, A, Tp, Status),!);
    (release_not_possible(R, A, Tp, Status),!).          (PRO 73)
release_completed(R, A, Tp, Status):-
```

[1] i.e the enabling state of the activity has enabled or committed as status

```
    activity_history(R, Act_List, Tp), member(A, Act_List),
    Status = completed.                                        (PRO 74)
release_committed(R, A, Tp, Status):-
    has_current_activity(R, Tp, Act_list),
    member_of(A, Act_list),
    Status = committed.                                        (PRO 75)
release_not_possible(R, A, Tp, Status):-
    (\+ in_commitment(R, A, Tp, Status);
    \+ in_current_activity(R, A, Tp, Status);
    Status = not_possible.                                     (PRO 76)
in_commitment(R, A, Tp, Status):-
    committed_to(R, A, S, Ti, Q, U).                           (PRO 77)
in_current_activity(R, A, Tp, Status):-
    has_current_activity(R, A, Tp, Act_list),
    member_of(A, Act_list).                                    (PRO 78)
```

Example:

The use of (EX 94) when we need to check what is the status of the release action by *assemble_hand* activity on *assembly_area_1* resource:

```
?- release_state(pro_assemble_hand, tp3, Status).    (EX 94)
```

would return the variable **Status = completed**. That is due to the fact that the resource 'assembly area 1' was committed to the activity in a period, *pd1*, which occurs prior to the time of the check, *tp3*.

On the other, if the same state is checked about the release action status at a different time point, *tp2*, by using:

```
?- release_state(pro_assemble_hand, tp2, Status).    (EX 95)
```

would return the variable **Status = committed.** This is due to the resource still being committed to the activity at the time of the check. In other words, the activity *assemble hand* is in the current activity list of *assembly area 1* resource[1].

[1] by satisfying the goal of has_current_activity axiom (PRO 46).

4.3.3 produce status: produce_state(State_ID, Tp, Status)

Definition:

A produce state in TOVE specifies that a resource, that did not exist prior to the performance of an activity, has been produced. The produce axiom would return the status of the produce state. Recall there are five possible status:

- **completed**: signifies that the resource has already been created.
- **not_possible**: signifies that a resource could not be produced because the conditions of the activity's enabling state could not be satisfied[1].
- **possible**: signifies that a resource could be produced because the conditions of the activity's enabling state could be satisfied[2].
- **committed**: signifies that a resource is committed to be produced. This implies that the conditions of the activity's enabling state are met but the activity producing the resource is scheduled to start later.
- **enabled**: signifies that the resource is currently being produced by the activity.

Semantics:

The produce state is **completed** if:

- the resource is in the resource's activity history list.

$$(\forall r, a)\ (\exists tp)\ produce\_completed(r, a, tp) \equiv (\exists state, act\_list)\ activity\_history(a, act\_list, tp) \wedge member\_of(a, act\_list) \qquad \text{(FOL 51)}$$

The produce state is **not possible** if:

- the use or consume enabling states of the activity are not possible.

$$(\forall r, a)\ (\exists tp)\ produce(r, a, tp) \equiv \exists(s_2, s_3)$$

$$((use(s_2, a) \wedge uses(s_2, r) \wedge enabling\_state(s_2, tp, not\_possible)\ ) \vee$$

$$(consume(s_3, a) \wedge consumes(s_3, r) \wedge enabling\_state(s_3, tp, not\_possible))) \qquad \text{(FOL 52)}$$

The produce state is **possible** if:

- the use and consume enabling states of the activity are possible.

$$(\forall r, a)\ (\exists tp)\ produce\_possible(r, a, tp) \equiv \forall s_2\ \forall s_3$$

[1] not included as the definition does not depend on resource ontology

[2] similar footnote 1.

$$((use(s_2, a) \wedge uses(s_2, r) \supset enabling\_state(s_2, tp, possible)) \wedge$$

$$(consume(s_3, a) \wedge consumes(s_3, r) \supset enabling\_state(s_3, possible))) \qquad \text{(FOL 53)}$$

The produce state is **committed** if:

- the activity causing the produce state is committed,
- *Tp* is before *Ti* and
- the activity is not executing.

$$(\forall\, r, a)\ (\exists\, tp)\ produce\_committed(r, a, tp) \equiv (\exists state, ti, q, u, act\_list)$$

$$(committed\_to(r, a, s, ti, q, u) \wedge after(ti, tp) \wedge$$

$$has\_current\_activity(r, act\_list, tp) \wedge \neg\ member\_of(a, act\_list) \qquad \text{(FOL 54)}$$

The produce state is **enabled** if:

- if the activity causing the produce state is committed and
- the activity is in the resource's current activity list[1].

$$(\forall\, r, a)\ (\exists\, tp)\ produce\_enabled(r, a, tp) \equiv (\exists\ state, act\_list)\ has\_current\_activity(r, tp, act\_list) \wedge$$

$$member\_of(a, act\_list) \qquad \text{(FOL 55)}$$

Implementation:

The term is defined with three arguments:

- **State_ID**: specifies the ID of the state
- **Tp**: specifies time point of the check
- **Status**: is the argument through which the status is returned.

```
produce_state(State_ID, Tp, Status):-
    (produce(A, State_ID), produces(State_ID, R)),
    (produce_completed(R, A, Tp, Status),!);
    (produce_committed(R, A, Tp, Status),!);
    (produce_enabled(R, A, Tp, Status),!);
    (produce_possible(R, A, Tp, Status),!);
    (produce_not_possible(R, A, Tp, Status),!).          (PRO 79)
produce_completed(R, A, Tp, Status):-
```

[1] i.e the activity is executing

```
    activity_history(R, Act_list, Tp),
    member_of(A, Act_list).                                   (PRO 80)
produce_not_possible(R, A, Tp, Status):-
    ((use(S2, A), uses(S2, R),
    enabling_state(S2, Tp, not_possible)) |
    (consume(S, A), consumes(S3, R),
    enabling_state(S3, Tp, not_possible)),
    Status = not_possible, fail.                              (PRO 81)
produce_possible(R, A, Tp, Status):-
    ((use(S2, A), uses(S2, R),
    enabling_state(S2, Tp, possible))|
    (consume(S3, A), consumes(S3, R),
    enabling_state(S3, Tp, not_possible)),
    Status = possible, fail.                                  (PRO 82)
produce_committed(R, A, Tp, Status):-
    committed_to(R, A, S, Ti, Q, U),
    time_period(Ti, ST, ET, Min, Dur, Max), Tp <= ST,
    Status = committed, \+activity_executing(R, A, TP),
    fail.                                                     (PRO 83)
activity_executing(R, A, Tp):-
    has_current_activity(R, Act_list, Tp),
    member_of(A, Act_list).                                   (PRO 84)
enabled(R, A, Tp, Status):-
    has_current_activity(R, A, Tp, Act_list),
    member_of(A, Act_list),
    Status = enabled, fail.                                   (PRO 85)
```

Example:

The use of produce axiom when we need to check what is the status of the produce action by *assemble_hand* activity on *assembly_area_1* resource:

```
    ?- produce_state(pro_assemble_hand, tp3, Status).    (EX 96)
```

would return the variable **Status = completed**. That is because the goals of *produce completed* axiom are satisfied. This is because the *assemble_hand* activity was committed at the time of the check (Tp) and is after the time of commitment of the activity, which produces the *hand_assembly*.

4.4 Conclusion

What is presented in this chapter is ontological terms required for modelling enterprise resources in a manufacturing environment. These terms have the characteristics of being generic and sharable across different enterprise applications. Hence enabling the share of information and coordination of activities.

The approach to accomplish sharability and reusability, is first to define a set of terms (ontology) using first order logic (*FOL*) and implemented in Prolog[1]. The ontology is stratified where the definition of each term is dependent on the definition of previous ones. The ontology consists of a number of ground terms (assertion) on top of which more complex terms are defined. First order logic is used because of its expressive and declarative capability. The rationale behind that approach is that with the ontology defined in FOL and applied in a theorem prover, the ontology could have the deductive capability to answer queries. Achieving that would give the model a common sense reasoning about the behavior of a world. Accordingly, the ontology provides a general descriptive language reason about the world.

The complexity of planning and scheduling is determined by the degree to which activities contend for resources. Planning involves selecting and sequencing activities to achieve a goal while scheduling involves assigning resources to activities over a time interval so that to obey temporal constraints and capacity constraints of shared resources. Accordingly, reasoning about resources is a critical component of the functionality of such systems. Ultimately, planning and scheduling systems have to be able to reason about availability and allocation of shared resources to activities which requires the ability to reason about the properties of resources when used or consumed by an activity.

[1] Refer to Goal and Objectives in chapter 1.

CHAPTER 5 *Capacity Recognition*

This chapter discusses the capacity recognition process in TOVE. This process includes a capacity recognition process that mainly addresses the issue of defining the capacity of a resource at a certain time point. Computational results are also presented.

5.1 Introduction

It is evident that constraints play an important role in scheduling activities; specifically, capacity constraint(s)/condition(s) should be the cornerstone of any scheduling activity since an activity (job)can use or consume a resource if the resource is available and can satisfy the requirement of the job. In other words the execution of an activity depends on the resource having the required *capacity* to fullfil the activity's requirement.

In TOVE, capacity is state dependent implying it is dependent on the activities already or wanting to use/consume the resource. As mentioned before[1], capacity in TOVE is defined as being the maximum set of activities that can simultaneously use or consume a resource at a specific time. Given a set of activities that require a resource, the capacity of the resource is defined in terms of a set of activities that can be supported by the resource without violating any of the capacity constraints. In the case where the activities requiring and using/consuming the resource are homogenous[2], then capacity of a resource is reducible to a number indicating the number of activities that the resource can support (could be allocated to). However in heterogenous activities case, the problem is reducible to a single machine

[1] in chapter four

[2] i.e require same amount over the same time period

sequencing problem where the machine can support multiple activities simultaneously.

What is presented in this chapter is the:

- definition of the different units of capacity,
- definition of the different categories of the capacity recognition process,
- definition of when activities are considered homogenous and when they are not.
- the formulation of the problem in Integer Programming to compare the output with the output resulted from applying the defined heuristic.
- definition of the sequencing heuristic that could be applied for the different categories.
- experimental results

5.2 Background and notations

Scheduling and sequencing problems in general are NP-complete/hard problems. For instance $(n / 3 / F / C_{\max})$[1] problem is NP-complete/hard [French 87]. A rudimentary problem in scheduling is the sequencing of *n* jobs using a single resource (facility) with the objective of minimizing the number of tardy jobs. An activity is described in terms of:

- *earliest starting time* (st_j): the earliest time point at which the activity can start (i.e *ready time*),
- *processing time* (p_j): amount of processing time required,
- *due date* (d_j): the completion time of the activity (i.e *latest end time*),
- *capacity requirements* (c_j) of each activity.

The above characteristics are all represented as integer numbers. Activities are performed when they are ready (i.e time $\geq st_j$) without allowing preemption of activities that are currently in progress. An activity is considered *tardy (late)* when it finishes at a time which is greater than its due date, while an activity is considered *on time* if it finishes at a time less than or equal to its due date. Tardy activities (jobs) are assumed to have equal penalties. Furthermore no precedence is assumed between different activities.

[1] Three machine, n-jobs, flow-shop problem where the aim is minimize the make-span

Each machine (resource), on the other hand, is described in terms of the total number of capacity units it contains[1]. Accordingly a resource can be used/consumed by more than one job depending on the number of capacity units it contains and the requirements of different activities that are to use/consume the resource. A resource can be used by an activity when ever the activity is ready and the resource has the capacity.

The problem is represented in the literature as $(n / 1 / r_j, p_j, c_j / \Sigma U_i)$[2] with the objective of minimizing the number of tardy jobs. A special case of this problem is when the value of c_j is restricted to 1[3], $1 / 1 / r_j, p_j / \Sigma U_i$. This special case is proven to be an NP-Hard problem as shown by [Lenstra et al 77] and [Graham et al 79]. Accordingly the $n / 1 / r_j, p_j, c_j / \Sigma U_i$ is NP-hard too.

Furthermore there exist other references proving other instances to be NP-hard [Baker 74] [Garey et al 79]:

- Sequencing to minimize tardy jobs with release times (ready times) and deadlines for a one machine is NP-hard.
- Sequencing to minimize tardy jobs for a one machine is NP-Hard. Jobs are characterized by having a due date and ready time.

There exists an algorithm due to Moore [Moore 68] that finds the solution of $n / 1 / / \Sigma U_i$ in polynomial time. In other words the problem is solvable in polynomial time in the case where all jobs have equal ready (release) times and jobs can be processed one at a time. Activities in Moore's algorithm are sequenced in ascending order due to their respective due dates and if the addition of job j results in a job being late, the scheduled job with the largest processing time is marked as late and removed from the sequence.

5.3 Units of Capacity

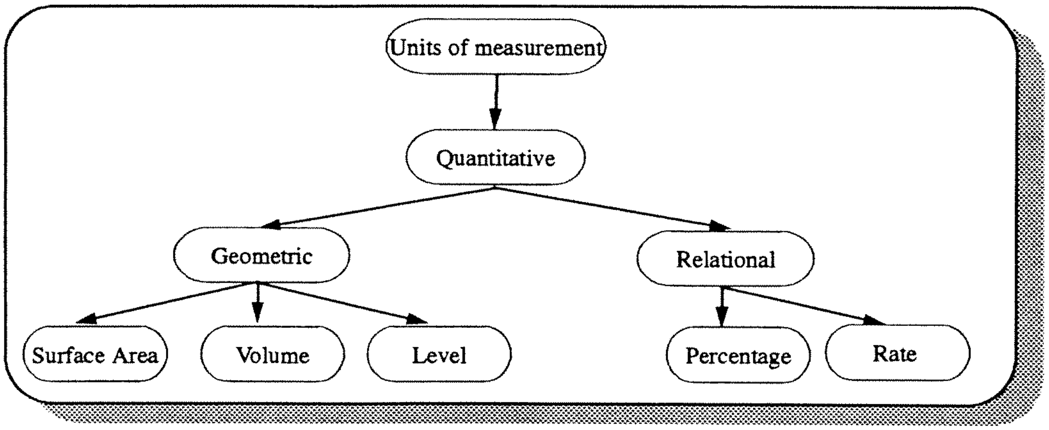
Before getting into the capacity recognition process, units of capacity applied to different resources are defined.

[1] That is in the capacity recognition context. In chapter three a general frame work of resource description is presented.

[2] single machine, n-jobs problem with different ready and processing times. The aim is minimize the number of tardy jobs.

[3] hence making the a single machine and one job problem

FIGURE 47 Taxonomy of Units of measurement - capacity recognition perspective[1]



In chapter four an overall taxonomy of unit of measurement is presented. Figure 47 presents a portion of the taxonomy showing the "quantitative" units used in capacity recognition process. There are basically two classifications of capacity units, with sub types, that will be used as a means of measurement: geometric and relational units.

5.3.1 Geometric units:

These are units with which the capacity of a resource could be solely described. These units utilize rectilinear or curvilinear motifs. This class is further classified into:

1. **Surface Area:** By sub dividing the surface area of a resource to number of equal *unit rectangles*, in inch^2 for example.
2. **Volume:** The resource is divided into an equal number of *unit blocks*, say in inch^3.
3. **Level:** The means of measurement in this case is the physical level of the resource.

5.3.2 Relational units:

This class of unit of capacity is used when the capacity is to be described in terms of *geometric* unit measurements. For example:

1. **Rate of consumption:** It is the rate by which an activity consumes a resource.

[1] Reference to the units of taxonomy in chapter four

2. **Percentage:** It is the percentage of usage/consumption of a resource by an activity.

The rate of consumption and the percentage of usage/consumption could be defined in terms of volume, level or surface area units of capacity (amount description units). The choice of using the above units not only depends on the resource but also on the activity or activities using/consuming the resource.

5.4 Indicative vs. Non-indicative level of recognition

In TOVE there are two scenarios of capacity description[1]:

1. **Indicative level of recognition:** where the capacity of a resource is solely determined through one unit of capacity.
2. **Non-indicative level of recognition:** where the capacity is determined through a number of units and/or other related factors such as the resources's layout.

Before explaining the application of the levels of capacity recognition, let us consider these two cases:

Pizza Oven case:

The unit of capacity in a pizza oven could be *surface area*, or *rectangles, as* the primary aim is to check whether a pizza could fit into the oven. For example there are three types of activities that could use the resource 'oven'.

1. 'Bake Large Pizza'.
2. 'Bake Medium Pizza'.
3. 'Bake Small Pizza'.

Each of the above activities requires different capacity, surface area, of the oven depending on the size of the pizza. Accordingly, the oven could support an activity if the number of unoccupied units and adjacent rectangles will satisfy the activity's requirement.

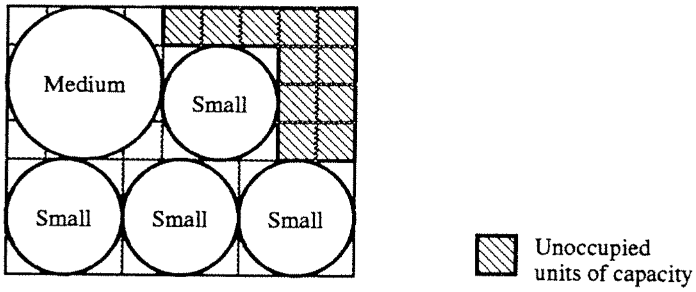
Water Tank case:

The unit of capacity in the water tank case could be the rate of consumption of the resource "*water*" by different activities at a specific time period.

[1] Figure 49 .

The difference between the two cases is that in the water tank case, the capacity is primarily defined in terms of the rate of consumption of water in the tank. However, in the pizza oven problem, the capacity of the resource is not solely defined by the number of unoccupied unit rectangles, which is a non-indicative capacity unit, but also by the layout of pizzas in the oven. On the other hand, the two cases are similar in the fact that the capacity of each resource is variable in a sense that the capacity is dependent on the activities supported by the resource. In the pizza oven case the capacity of the oven is dependent on the number and type of pizzas already supported by the oven. Similarly, the capacity recognition of water tank is dependent on the supported activities. The phenomenon of the capacity of a resource being state dependent is referred to as a variable capacity case.

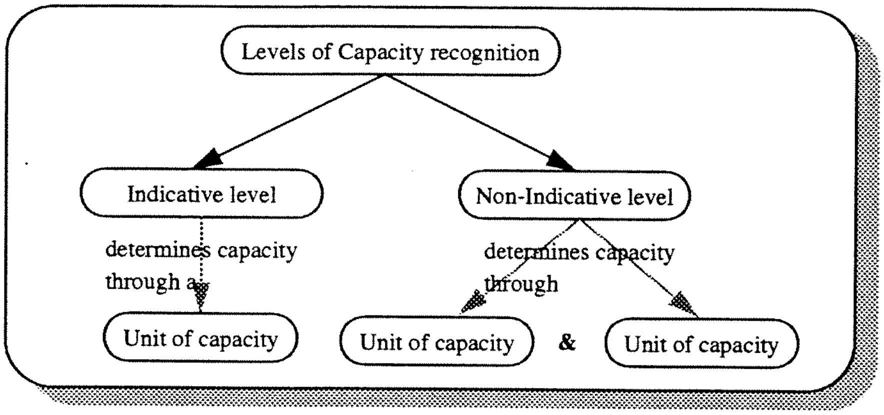
FIGURE 48 Pizza oven example



In figure 48 a top view of a pizza oven shows that five activities are currently using the oven occupying a number of unit rectangles. In this example a small pizza occupies nine unit rectangles while the medium one occupies sixteen unit rectangles. The unoccupied unit rectangles in the oven is eleven rectangles which could accommodate another small pizza. However this can not happen due to the layout restriction. The current layout of the unoccupied blocks does not permit another pizza to be accommodated into the oven.

Accordingly, the capacity of the pizza oven is not solely defined through the number of unoccupied unit rectangles but also with other levels such as the layout of the unoccupied unit rectangles. This is different from the case of the water tank capacity recognition where the capacity could be determined through only using the rate of usage by different activities. That is to say, the layout is not a factor in the water tank case as obviously liquids lack form.

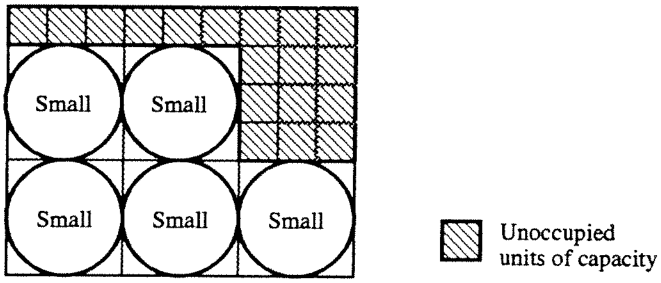
FIGURE 49 Indicative vs. Non-indicative levels of recognition


Levels of Capacity recognition
Indicative level
Non-Indicative level
determines capacity through a
Unit of capacity
determines capacity through
Unit of capacity
&
Unit of capacity

5.5 Homogenous vs. Heterogenous activities

Consider the case where a resource, such as the pizza oven example, that has a number of non-indicative levels of capacity recognition. A unit rectangle is non-indicative in capacity recognition, but the same unit could become indicative in the case where all activities using/consuming the resource are **homogenous**, as shown in figure 50.

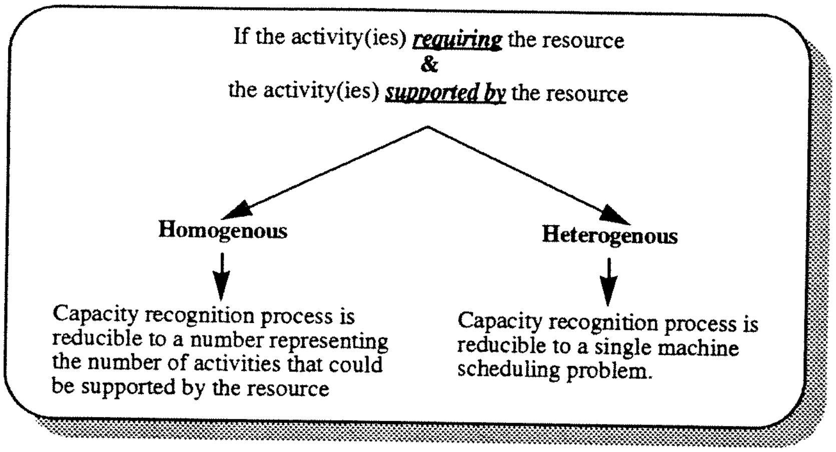
FIGURE 50 Usage of the oven by Homogenous activities



Activities (jobs) could be homogenous if they require equivalent amounts of the resource or processing or integral multiple thereof. In the pizza oven case, homogeneity specifies that each activity occupies the same space requirements (i.e unit rectangles). In this case, the capacity recognition is reducible to a number which represents the number of activities the resource can support. Here the layout of the oven is not a factor any more, in contrast to the case shown in figure 48, as the maximum number of activities the resource can support is predetermined. On the other hand **heterogenous activities** are characterized by having dissimilar requirements which increases the complexity of the problem making it NP-Hard. In this case, the capacity recognition process is reducible to a sequencing problem with an objective that could be to minimize the number of tardy jobs, average tardiness, average waiting time etc.

As shown in the figure 51 the capacity recognition process depends on whether the activities, requiring and supported by a resource, are homogenous or heterogenous.

FIGURE 51 Taxonomy of Capacity Recognition


If the activity(ies) requiring the resource
&
the activity(ies) supported by the resource
Homogenous
Heterogenous
Capacity recognition process is reducible to a number representing the number of activities that could be supported by the resource
Capacity recognition process is reducible to a single machine scheduling problem.

The homogenous case is addressed in chapter four and implemented in the Prolog environment. The heterogenous case is addressed by defining a heuristic for the sequencing of independent activities on a single machine. The output of the heuristic is a sequence of activities that could be supported by the resource.

5.6 Integer Programming Formulation

Efforts to solve a resource constraint scheduling problem using an integer programming formulation have been described in number of papers. Some of the attempts were to solve non-preemptive scheduling such as [Talbot 82] and [Patterson 74] or preemptive scheduling such as [Welglarz 77]. Other efforts were directed towards preemptive scheduling where activities are described by continuous performance speed-resource functions and discrete time-resource functions [Slowinski 80].

The capacity problem is viewed as being reducible to the non-preemptive scheduling for activities (jobs) with release times and due dates in a specified resource time window. The integer programming formulation is applied to the pizza oven problem for both the homogenous and heterogenous cases with the objective function being to minimize the number of tardy jobs in the time window. Three types of activities are to use the resource. Moreover, the time window (horizon) will be divided into discrete time slots to which jobs can be assigned to.

5.6.1 For Homogenous activities

TABLE 2 List of Variables

Variable	Description
x_{jt}	= 1; if job *j* is scheduled to start in time interval t^a = 0; otherwise
y_{jt}	= 1; if job *j* is still being processed in time interval *t* = 0; otherwise.

a. time horizon is divided on discrete basis

TABLE 3 Notations

Notation	Description
N	number of jobs (activities)
T	resource time window
st_j	earliest starting time of job *j* (release time)
et_j	latest ending time for job *j* (due date)
p_j	processing time of job *j*
C	total capacity of the resource
j	job number j =1...N
t	time intervals t = 0....T^a

a. time interval 0, for example, represents a time interval occurring between [0...1] time points.

Objective function:

As defined in table 2, x_{jt} specifies what is the time slot to which each job is assigned to start. The objective is to minimize the number of tardy jobs. This is achieved by maximizing the number of scheduled jobs in the specified time window.

$$Maximize \sum_{t=0}^{T} \sum_{j=1}^{N} x_{jt} \qquad \text{(EQ 1)}$$

Subject to:

A job has at most one start time.

$$\sum_{j=1}^{N} x_{jt} \leq 1 \qquad \text{(EQ 2)}$$

$$for\ j = 1 \ldots N$$

Each job can not be scheduled before the relative release time.

$$\sum_{t=0}^{stj-1} x_{jt} = 0 \qquad \text{(EQ 3)}$$

$$for\ j = 1 \ldots N$$

Each job can not be scheduled after the latest start time[1].

$$\sum_{t=etj-pj+1}^{T} x_{jt} = 0 \qquad \text{(EQ 4)}$$

$$for\ j = 1 \ldots N$$

In the case of homogenous jobs (activities) the capacity is reducible to a number which indicates the number of jobs that can simultaneously use the pizza oven. Accordingly, the maximum capacity denoted by *C*, indicates that the number of jobs that could start at a specific time point and the number of jobs that can still be using/consuming a resource are less than or equal to *C*.

$$\sum_{j=1}^{N} (x_{jt} + y_{jt}) \leq C \qquad \text{(EQ 5)}$$

$$for\ t = 1 \ldots T$$

[1] latest start time = $et_j - p_j$

The problem has been defined as being a non-preemptive one. Hence continuity of the processing times of the activities, supported by a resource, should be ensured and that is done through the continuity constraints. As defined before y_{jt} denotes that job *j* is still in process at that time. The continuity issue is ensured through defining a series of constraints as specified in equation (EQ 6) through (EQ 7). The number of these constraints depends on the processing time of the job, denoted by *pj*.

$$y_{j(t+1)} - x_{jt} = 0 \tag{EQ 6}$$

$$\bullet$$

$$\bullet$$

$$\bullet$$

$$y_{j(t + (pj-1))} - x_{jt} = 0 \tag{EQ 7}$$

Hence, the series of constraints for each job must be repeated over all possible starting times of the job.

$$for \;\; t = st_j \ldots (et_j - p_j)$$

The series of equations (EQ 6) through (EQ 7) are to be formulated for each job.

$$for j = 1 \ldots N$$

Objective function revisited:

With the current problem formulation, y_{jt} variables that are not included in the continuity constraints will not be constrained to be assigned the value of zero. One solution is to add more constraints to the formulation or to add all y_{jt} variables in the objective function[1].

$$Maximize \sum_{t=0}^{T} \sum_{j=1}^{N} (x_{jt} - 0.1 y_{jt}) \tag{EQ 8}$$

y_{jt} variables are placed in the objective function with a negative sign in order to force all unnecessary y_{jt}'s to zero. A multiplier of 0.1 is used so that the weight y_{jt} is reduced in the objective function and will not cancel out the weight of x_{jt}. Without the use of the multiplier, the problem solver will neglect solving for y_{jt} as the objective function is not affected by them, for the case where the processing time of the job is two time units. This is due to the presence of the continuity constraint

[1] i.e $\sum^{T} y_{jt} = 0 \quad forj = 1 \ldots N$ constraints will not be included. The number of these constraints is equal to the number of y_{jt} variables that are included in the objective function.

($y_{j(t+1)}$ - x_{jt} = 0) which will cancel the effect of the corresponding x_{jt} variable in the objective function. In the case were the processing time of a job is greater than two time units, then the job will not be scheduled as for each x_{jt} variable there will be at least two corresponding y_{jt} variables hence decreasing the value of the objective function.

$$x_{jt},\ y_{jt}\ldots Integers \tag{EQ 9}$$

$$0 \le x_{jt} \le 1 \tag{EQ 10}$$

$$0 \le y_{jt} \le 1 \tag{EQ 11}$$

$$-\infty \le st_j \le \infty \tag{EQ 12}$$

$$-\infty \le etj \le \infty \tag{EQ 13}$$

$$-\infty \le N \le \infty \tag{EQ 14}$$

$$-\infty \le T \le \infty \tag{EQ 15}$$

5.6.2 For Heterogenous activities

In the heterogenous case, the unit rectangles capacity approach will be used. In other words, the surface area of the oven will be divided into unit rectangles and each job (activity) will have a requirement in terms of number of unit rectangles needed. Hence the capacity constraint, (EQ 5), will have to be reformulated but the rest of the formulation of the homogenous case will be used as is. Accordingly each activity of the pizza baking activities will have a predefined capacity requirements specified in terms of unit of rectangles. Reference to the pizza oven example:

- baking a small pizza will require nine units rectangle of capacity.
- baking a medium pizza will require sixteen units rectangle of capacity.
- baking a large pizza will require twenty unit rectangle of capacity.

The main issue here is how to divide the surface area of the oven in such a way that the situation expressed in figure 48 would be avoided.

TABLE 4 List of Variables

Variable	Description
x_{jt}	= 1; if job *j* is scheduled to start in time interval *t* = 0; otherwise
y_{jt}	= 1; if job *j* is still being processed in time interval *t* = 0; otherwise.

TABLE 5 Notations

Notation	Description
N	number of jobs (activities)
T	resource time window
st_j	earliest starting time of job *j* (release time)
et_j	latest ending time for job *j* (due date)
p_j	processing time of job *j*
r_j	requirements of job *j* in terms of unit rectangles
C	capacity of the resource in terms of unit rectangles. It is the total number of sub divisions
j	job number j = 1...N
t	time points t = 0....T[a]

[a] time interval 0, for example, represents a time interval occurring between [0...1] time points.

In this formulation a new notation is added, r_j, which specifies the needed unit rectangles for job *j*. Hence (EQ 5) will have to be changed to add the new notation.

$$\sum_{j=1}^{N} r_j (x_{jt} + y_{jt}) \leq C \qquad \text{(EQ 16)}$$

$$for \ t = 1 \ldots T$$

5.7 Heuristic approach

Sequencing theory for a single machine has been addressed by a number of research. For instance, Moore has devised an algorithm for sequencing independent jobs on a single machine [French 87] [Nahmias 89]. Also, Fisher and Jaikmar [Fisher 78] used the same approach to derive a scheduling algorithm for the space shuttle. The common dominator between the two is that the objective is to minimize the number of tardy jobs and all tardy jobs have equal penalties no matter how late each job is.

It has already been mentioned that capacity recognition in TOVE varies according to two cases:

- homogenous activities using/consuming the resource and in that case the process is reducible to a number which represents the number of activities that can use/consume the resource at a specified time point.
- heterogenous activities using the resource and in that case the process is reducible to a scheduling (sequencing) problem of all activities that requires the resource at a specific time point.

As a demonstration, the pizza oven problem is going to be solved with three types of activities that require the oven (resource):

1. 'Bake Large Pizza'.
2. 'Bake Medium Pizza'.
3. 'Bake Small Pizza'.

Each activity requires different capacity (surface area) requirements, ready times, processing times and due dates. The objective is to minimize the number of tardy jobs as all tardy jobs have equal tardy penalties. The underlying assumption in the heuristic[1] is that the total units of capacity of the oven is configured in a way that no situation could occur were the layout of the pizzas in the oven is an element in the capacity recognition process (i.e the situation described in figure 48).

5.7.1 Heuristic:

The starting point of the heuristic is borrowed from Moore's algorithm. Activities in Moore's algorithm are sequenced in ascending order due to their respective due dates and if the addition of job *j* results in a job being late, the scheduled job with

[1] The same as the underlying assumption in the Integer Programming formulation.

the largest processing time is marked as late and removed from the sequence. In the proposed heuristic, activities that could cause any of the constraint violations, are temporarily considered tardy.

The applied approach is first to build an initial sequence that could contain capacity violations (phase 1), then incrementally enhance the sequence so that a capacity violation-free sequence could be produce (phase 2-4).

Phase 1: (see figure 53)

The objective of this phase is to build an initial sequence without checking for capacity violation. Second, find the jobs that are candidates for being the cause of capacity violations and temporarily discard them from the EDD job sequence.

1. Find all activities that are to use a resource in a specified resource time window.
2. Sequence the activities (jobs) according to a criterion. This criterion could be: Earliest Due Date (EDD), Shortest Processing Time (SPT), Critical Ratio (CR) etc. depending on the objective. Since the objective is to minimize the number of tardy jobs, therefore the used criterion is EDD.
3. Select the first job in the sequence.
4. Schedule the chosen job without checking for capacity violation. The job is scheduled according to its earliest starting time to according to its latest start time.
5. Update the job sequence (EDD sequence)[1].
6. Is that the last job in the sequence? If **NO** go to **step 3** otherwise continue.
7. For each time point in the resource time window, check the capacity constraint.
8. Any capacity constraint violation? If **NO** go to **step 12** in phase 2, otherwise continue.
9. Find the first capacity violation.
10. Choose a job in the current sequence and temporarily discard it from the current sequence. The choice criteria is:

- the job with the highest capacity requirements **OR**
- the job with the longest processing time **AND** the job with the highest capacity requirements[2] **OR**

[1] i.e remove the scheduled job from the sequence

- among the jobs currently occupying the resource, discard the job with the longest processing time.

11. Go to **step 7.**

Phase 2: (see figure 54)

The objective of this phase is to try to schedule the jobs, that were discarded from phase 1, so that to meet their respective due date.

12. Any discarded jobs? If **NO** then go to **step 22**, otherwise continue.
13. Find the end time of the last scheduled job in phase 1 (milestone). (figure 52)
14. For all discarded jobs calculate the Critical Ratio (CR), where

$$\mathbf{CR} = p_j / (et_j - \text{milestone}) \qquad \text{(EQ 17)}$$

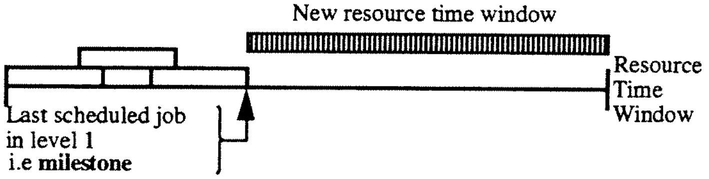
p_j → Processing time of job *j*

et_j → end time of job *j*

In other words, CR is just the ratio between the processing time and the available time for the job before being tardy.

15. Sequence the jobs with 0 < CR <= 1, in descending order according to the CR value.
16. Choose the first job from the sequence, and try to schedule it starting from the milestone.
17. Any more jobs in the CR sequence? If **Yes** go to 16, otherwise continue to phase 3.

FIGURE 52 Milestone



Phase 3: (see figure 55)

[2] Jobs are discarded according to their respective processing time. Jobs with the longest processing times are the ones that are discarded. The highest capacity requirement criterion is used whenever ties occur in terms of jobs to be discarded.

For the jobs with CR <= 0 and CR > 1, try to schedule them before the milestone so that they could meet their respective due date.

18. Any unscheduled jobs? If **No** then go to **END**, otherwise continue.
19. Try to schedule the job fulfilling both the capacity and temporal constraints.

Phase 4: (see figure 56)

If there are any jobs unscheduled, then these jobs are going to be scheduled, as tardy jobs after the scheduled job in the sequence.

20. No unscheduled jobs? If **Yes** go to **22**, otherwise continue.
21. schedule the jobs after the last scheduled jobs.
22. **END**.

FIGURE 53 Sequencing Heuristic: Phase 1

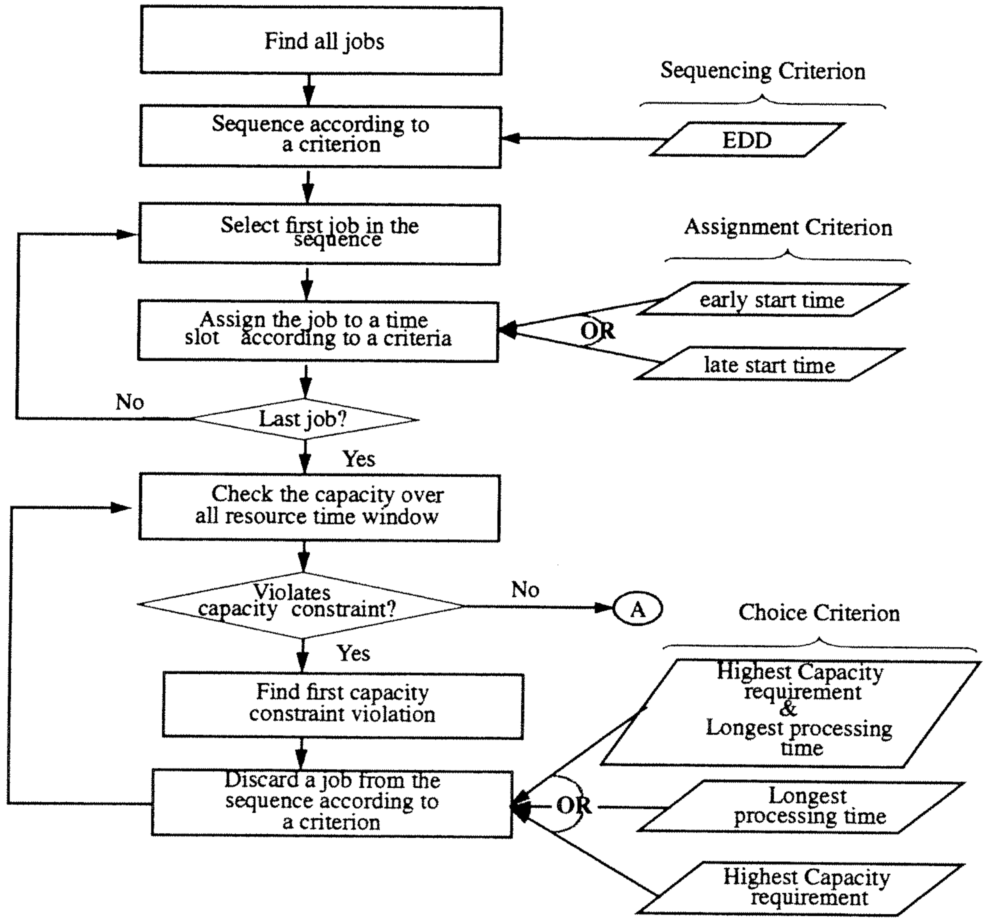

Find all jobs
Sequencing Criterion
Sequence according to a criterion
EDD
Select first job in the sequence
Assignment Criterion
early start time
Assign the job to a time slot according to a criteria
OR
late start time
No
Last job?
Yes
Check the capacity over all resource time window
Violates capacity constraint?
No
A
Choice Criterion
Yes
Find first capacity constraint violation
Highest Capacity requirement & Longest processing time
Discard a job from the sequence according to a criterion
OR
Longest processing time
Highest Capacity requirement

FIGURE 54 Sequencing Heuristic: Phase 2

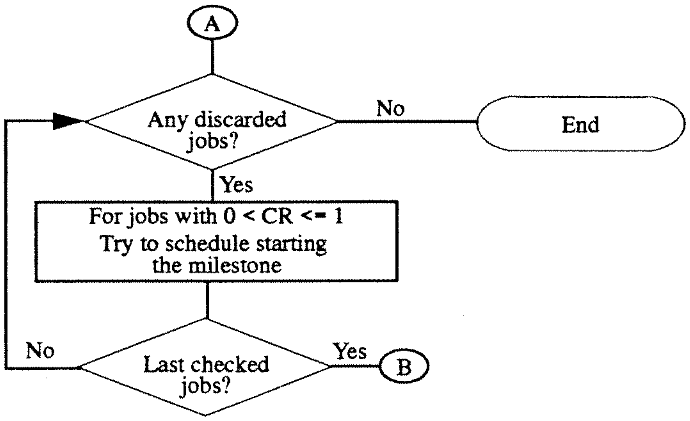

A
Any discarded jobs?
No
End
Yes
For jobs with 0 < CR <= 1
Try to schedule starting the milestone
No
Last checked jobs?
Yes
B

FIGURE 55 Sequencing Heuristic: Phase 3

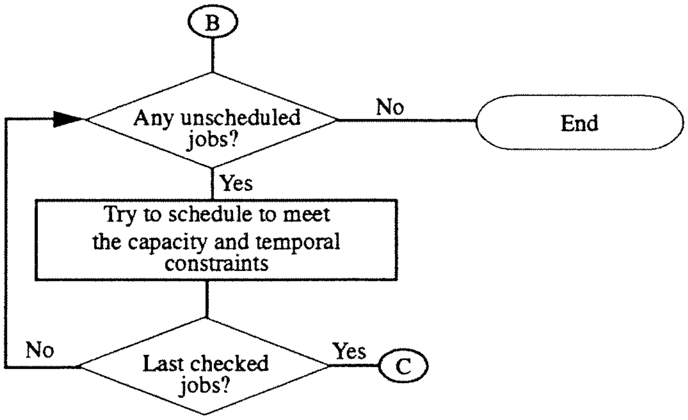
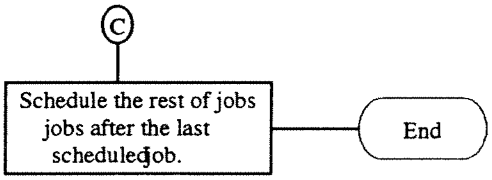

B
Any unscheduled jobs?
No
End
Yes
Try to schedule to meet the capacity and temporal constraints
No
Last checked jobs?
Yes
C

FIGURE 56 Sequencing Heuristic: Phase 4



5.8 Problem generation

The four versions of the heuristic were tested over two sets of data; in the first set (**Set A**), the starting times were based on Poisson distribution. Two values of lambda [λ] (inter-arrival times between jobs) were used: 3.75 and 7.5 minutes.

These values are based on actual data from McDonald's corporate store located in Scarborough[1]. Lambda = 3.75 minutes is for the dinner period (5:30 pm to 8:00 pm) while lambda = 7.5 minutes is for the lunch period (11:30 am to 2:30 pm). The processing times, end times and the capacity requirements of each job are assumed to be independent of the starting times and were drawn from a uniform distribution.

The second set (**Set B**) of data consists of 110 test problems which were specially designed to explore the performance of the heuristic by varying the problem size and the capacity requirements while keeping the starting processing time constant. Seven of these test problems have a known optimal solution obtained through using CPLEX mixed-integer programming optimizer. The rest of the test problems are used to compare the four versions of the heuristic.

5.9 Computational results

Four versions of the heuristic were tested against the two sets of data. The four heuristics are compared on the basis of the number of best solutions generated (i.e the number of times that each heuristic generated the least number of tardy jobs). If a tie occurred between more than one version, then each tied version is counted as the generator of the best solution. The average solution times are presented for the second set of data only as the number of jobs is controlled while for the first set, the number of jobs can not be controlled; this is because the data generated is based on a Poisson distribution. The heuristic was implemented in the C programming language.

Four versions of the heuristic were tested for the purpose of experimenting on the different choice criteria in the heuristic. In the first phase of the heuristic, the assignment of jobs to time slots is based on their:

- earliest start time
- latest start time

Moreover in the case when having capacity violations, the job's discarding criterion could be either:

- highest capacity requirement among the activities at the time of violation or

[1] Reference to Yan, Gary, Corporate store manager, McDonald's

- highest capacity requirement and the longest processing time among the activities at the time of violation.

The four versions of the heuristic are presented in table 6.

TABLE 6 Four versions of the heuristic

Version	Assignment of jobs	Criteria of discarding jobs
ESP	Earliest start time	Longest processing time
ESPR	Earliest start time	Longest processing time & Highest capacity requirement
LSP	Latest start time	Longest processing time
LSPR	Latest start time	Longest processing time & Highest capacity requirement

5.9.1 Testing against set A data

Table 7 presents the results of comparing the four versions of the heuristic based on the number of time each heuristic generated the best solution. This set contained 65 test problems.

TABLE 7 Performance of the four versions with respect to set A

Heuristic	Frequency of generating best solutions
ESPR	58
ESP	1
LSPR	63
LSP	0

The above table shows that the **LSPR** version generated the best solutions, then followed by the **ESPR** version.

5.9.2 Testing against set B data

Results of testing against 110 problems are:

TABLE 8 Performance of the four versions with respect to set B

Heuristic	Frequency of generating best solutions
ESPR	23
ESP	41
LSPR	62
LSP	39

Accordingly, **LSPR** version generated the highest number of best solutions. As for the solution times, they are almost equal as presented in the following table.

TABLE 9 Solution times of the four heuristic

Heuristic	Solution times in seconds	
	for 93 jobs	for 136 jobs
ESPR	0.3	0.7
ESP	0.6	0.9
LSPR	0.3	0.6
LSP	0.5	0.8

Moreover, the four versions were first tested against problems with known optimal solution (see appendix D). For these seven test problems, *LSP* and *LSPR*[1] versions generated the corresponding optimal solution.

Based on the above results, the different versions sorted by the number of tardy jobs they generate, starting with the version with the least number of tardy jobs, are:

1. LSPR
2. ESPR
3. ESP
4. LSP

[1] *LSPR* version did not generate the optima solution only in one problem out of the seven problems tested.

The preferable heuristic sequences the jobs according to latest start time, and discards jobs with the longest processing time and with the highest capacity requirement. Jobs with the longest processing time are discarded at time instances were capacity violations occur. Furthermore ties are broken by discarding the job that requires the highest capacity of the resource.

5.9.3 Integer programming approach

The integer programming formalism was also tested for three problems for problem sizes of 10, 20 and 40 jobs. It should be mentioned that problems with size 20 and 40 were created a duplicate of the 10 jobs problem[1]. That is to say, the 20 jobs problem is created from two problems of size 10[2]. The formulation was tested using CPLEX mixed integer optimizer developed by CPLEX Optimization Inc. The same three problems were tested using ESP heuristic and the results are represented in table 10.

TABLE 10 I.P solution vs. ESP heuristic version

	I.P solution		ESP heuristic solution	
Jobs	CPU Time (sec)	Tardy Jobs	Solution time (sec)	Tardy Jobs
10	0.28	1	0.0	3
20	22.63	2	0.0	6
40	4644.78 [a]	4	0.1	12

[a] actually it took more than 10 hours real time.

From the results, it is clear that the heuristic does not generate the optimal solution in these test cases. In the 10 job size problem the difference between the two approaches is two tardy jobs. However, the difference keeps on increasing linearly and that is mainly because of the way the 20 and 40 jobs problem were set up. As mentioned these problems are just a concatenation of a number of 10 size jobs problem. As expected the solution time of the IP increases exponentially while the heuristic solution time does not. For the 40 jobs problem, it was solved in 0.1 seconds, on average, using the heuristic while it took around an hour CPU time to solve the same problem in CPLEX. Actually it took around ten hours real time to solve the 40 jobs problem.

[1] this approach is only used for the testing of the IP.

[2] i.e the starting times of the 10 jobs size problem are incremented by a factor then appended to the original 10 jobs problem. The time horizon of the appended jobs overlaps with that of the original 10 jobs.

5.10 Conclusion

What is presented in this chapter is the capacity recognition process. The purpose of including this process is to give the TOVE the ability to set a capacitated schedule. In Tove, capacity of a resource is the maximum set of activities that a resource can support which makes the resource dependent on the activities being supported and the ones requiring the resource. The tackled problem is proved to be NP-hard in the strongest sense.

A capacitated modelling approach and a heuristic for resource allocation are presented. The capacity recognition process is reducible to a number in the case where all activities requiring and supported by the resource are homogenous. On the other hand, if the activities are heterogenous then the process is reducible to a single machine scheduling problem. A scheduling heuristic is suggested for the application in the heterogenous case and its performance is compared to that of I.P formalism.

With the capacity recognition process, that includes the resource ontology defined in chapter four, a planner/scheduler is able to reason about allocation of shared resources. That is achieved through reasoning whether a resource can support multiple activities or not, through use of resource ontology[1], and then the activities that require the resource are sequenced with the objective of minimizing the number of tardy jobs.

[1] e.g functional/physical divisible

CHAPTER 6 Competency Questions

This chapter presents a set of competency questions that should be answered by any manufacturing reference model. These questions are stratified so that each level refers to functionality supported by previous levels. Through the redesigning of the short shaft scenario, a list of competency questions is presented to demonstrate the use of the ontology in inter-disciplinary communication.

6.1 Introduction

In chapter one a number of criteria for the design and evaluation of an ontology are introduced: generality, competency, consistency, perspicuity, granularity, transformability, efficiencyc scalability and extensibility. The competency criterion is chosen as it defines the functional requirements of the model. The competency questions included in this chapter integrate questions that are addressed by the enterprise models and planning/scheduling efforts reviewed in chapter two in addition to new set of questions. As in [Fadel & Fox 94] the obvious way to demonstrate competency is to define a set of questions that can be answered by the ontology. The set of questions represents the basic accesses a problem solver makes to the representation.

Section 6.2 presents a scenario demonstrating the use of the ontology in answering queries in a collaborative engineering environment. It also contains questions required by the *resource manager* in the *Integrated Supply Chain Management* environment.

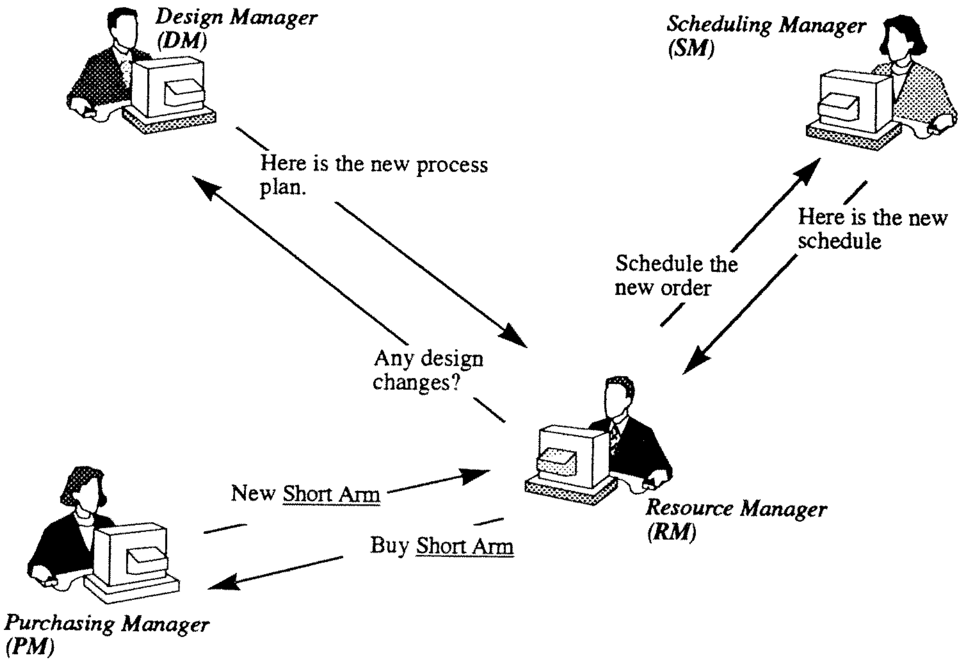
6.2 A scenario of inter-disciplinary communication

In this chapter a scenario in which communication among a product development team is presented as an example of competency questions that could be asked using the resource ontology. Team members are assumed to be from different disciplines working on the development of a product and hence need to coordinate their actions.

The scenario presented herein deals with the design of a clip reading lamp as presented in TOVE testbed. The specific case involves the process of redesigning a part of the clip reading lamp - short shaft. The clip base is assumed to be no longer available from supplier, hence the choice of a new clip base initiates a set of design changes in the short shaft design.

The outline of the scenario presented in figure 57, specifies that the scenario is initiated when the purchasing manager (*PM*) sends a message to the resource manager (*RM*) informing him of the unavailability of the needed clip base; purchase manager *(PM)* indicates that a similar item is available. The *RM* in turn sends a message to the design manager (*DM*) informing him of the changes, upon which the design of the short shaft is altered. The resulting modified design and the process plan of the shaft is returned to the *RM*. The *RM* asks the scheduling manager (*SM*) about the possibility of scheduling the production of the clip reading lamp with the modified short shaft design. Finally the *SM* informs the *RM* that production can be scheduled and a message to buy the clip base is relayed to the purchase manager.

FIGURE 57 An example of inter-disciplinary communication



The scenario presented in table 11 shows:

- the sender of the message
- the receiver, if any,
- message sent.

The presented scenario focuses on the manufacturing of the short shaft only and the respective activity cluster is presented in figure 58 and figure 59[1].

[1] It is assumed that TOVE factory has an order of two clip reading lamp and that the scenario is going to be on producing two modified short shaft.

FIGURE 58 activity cluster: fabricate short shaft

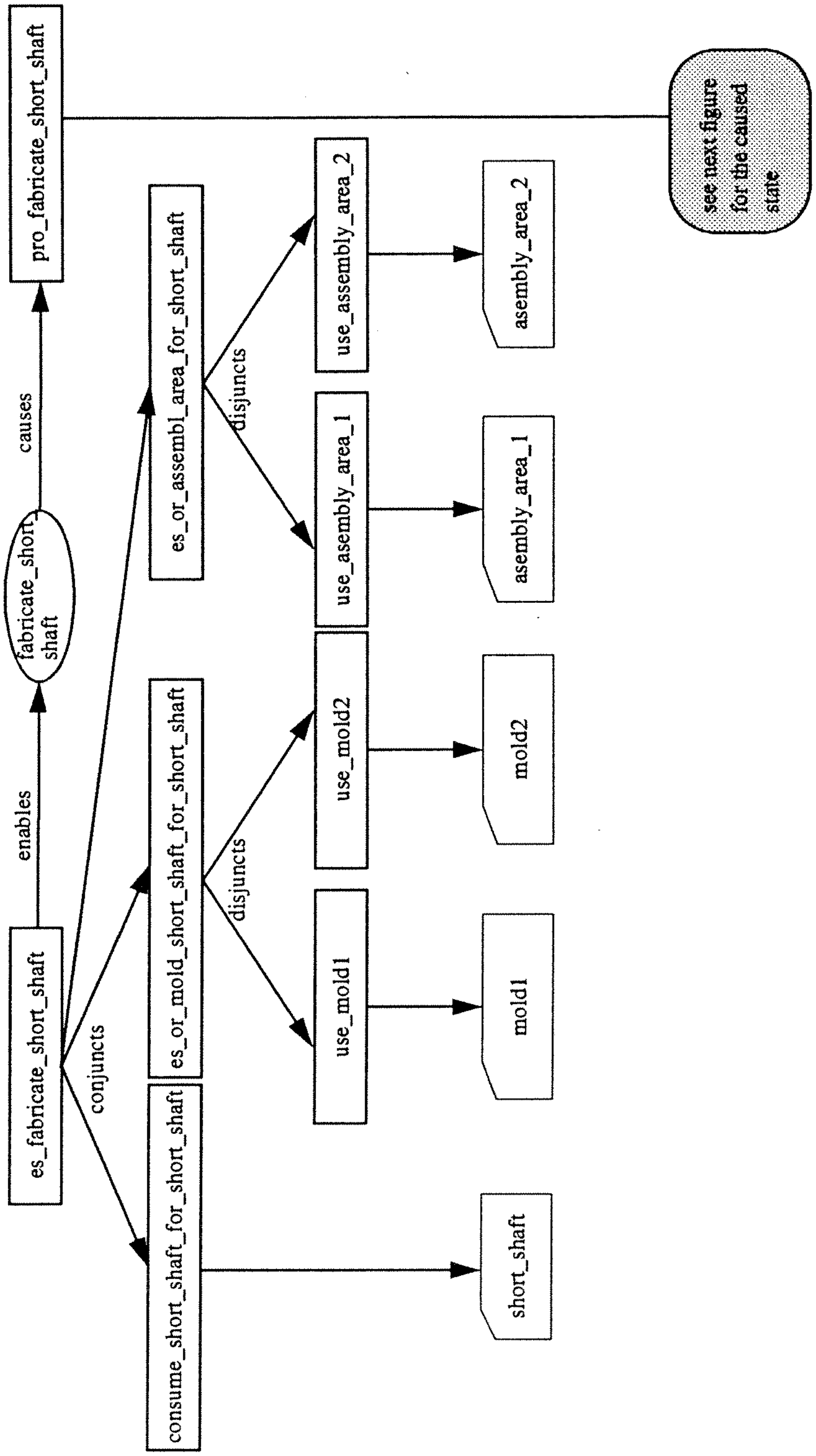
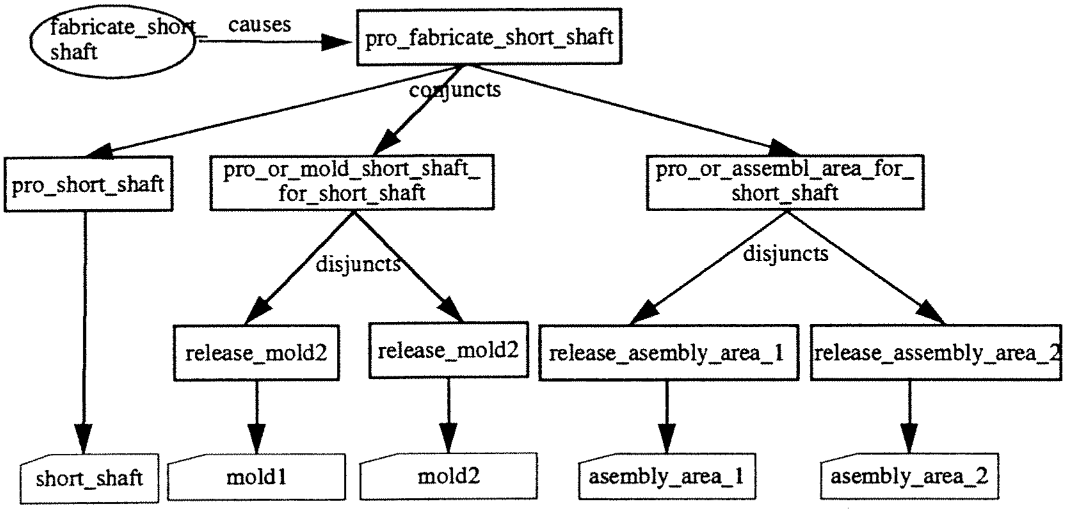


FIGURE 59 activity cluster: fabricate short shaft cont'd



Based on the scenario presented in figure 57, the highlight of the communication between departments is presented in table 11. This shows where the ontology is used in the process of sharing information between different departments.

TABLE 11 **Competency questions:** short shaft redesign scenario [*questions are highlighted*]

Sender	Receiver	Message	System's answer	Comment
PM	RM	New clip base		
RM	DM	New clip base, can it be used?		
DM	RM	Yes with the following process plan		
DM		?- rknown(short_shaft_2).	No	have to produce short_shaft_2 as this item with the specific was not produced before
RM	SM	Can the new order be scheduled?		
SM		?- rexist(mold1, Tp1). [a]	Yes	mold1 exists at Tp1.
SM		?- available_for(mold1, [A1,A2], Ti1,_).	[A1]	mold1 is available for only A1 activity.
SM		?- rexist(mold21, Tp2).	Yes	mold2 exists at Tp2.
SM		?- trend(mold1, Tp2, Result).	Result = increasing	mold1's capacity is increasing after Tp2.
SM		?- available_for(mold1, [A2], Ti2,_).	Yes	mold2 is available for A2 at Ti2?
SM		?- mobile(mold1), mobile(mold2).	Yes	mold1, mold2 are mobile.
SM		?- available_for(assembly_area_1, [A1], Ti,_).	Yes	assembly_area_1 is available for A1.
SM		?- available_for(assembly_area_1, [A2], Ti2,_).	Yes	assembly_area_2 is available for A2.
SM		commit mold1, mold2, assembly_area_1 & assembly_area_2		i.e assert committed_to

a.

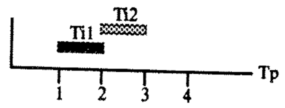


Table 11 presents the queries required to fulfill the order of the clip reading lamp while the coming section presents examples of such communication showing how the query is answered by the use of other ontology. For each query, the Prolog axiom presented[1]. This is followed by a list of terms and axioms required to answer the query.

☞ ☞ *How much of the resource* `short_shaft` *is available now?* (What is the available capacity of resource `short_shaft` at time tp1?)

```
|?- available_capacity(short_shaft, tp1, Amount, Unit).
  Requires
      |?- rp(short_shaft, tp1, Q, object).
```

Calculating the stock level

```
        Requires
            |?- rpl(short_shaft, tp1, Q, L, object).
            Q = 12
            L = 231;
            Q =2
            L = 323;
            no
```

12 objects are available at location 231 & 2 objects are available at location 323

```
      Q = 14
      Unit = object;
      no
      |?- total_committed(short_shaft, TQ,tp1, object).
```

Calculating the total commitment

```
        Requires
            |?- committed_to(short_shaft,A,S,Q,TP,object).
            A = fabricate_short_shaft_2
            S = es_fabricate_short_shaft_2
            Q = 2
            Tp = tp1;
            no
      TQ = 2;
      no
Amount = 12
Unit = object;
no
```

12 are the uncommitted objects of the resource

The hub of planning and a scheduling activities is to be able to reason about the availability of the resource. Reasoning about the availability can not be performed

[1] A query in Prolog is proceeded by "?-". A variable is represented as a capital letters.

unless there exists a representation of the requirements, nature of usage, commitment, constraints and resource amounts.

☞ ☞ *Is there enough capacity for the performance of "fabricate_short_shaft_1" and "fabricate_short_shaft_2" activities on the "mold1" resource in the next hour?* (Given a number of activities requiring the resource ***mold1***, what is the ***capacity*** of the resource during a time period?)[1]

```
|?- available_for(mold1, [fabricate_short_shaft_1,
fabricate_short_shaft_2], pd3, Result).
```

[1] *available_for* requires the availability of the following terms: functional divisible, physical divisible, unit of measurement, has current activity, consumption and use specification, total committed, resource point, simultaneous use restriction and time ontology.

Requires

```
|?- functional_division_of(R, mold1).
no.

|?- physical_division_of(R2, mold1).
R2 = rectangle_1;
no
```

division_of is used to reason about the sharability of the resource.(i.e used in divisible term)

```
|?- unit_of_measurement(mold1,Unit_ID,U,
    fabricate_short_shaft_1).
Unit_ID = rectangle_1
U = rectangle_1;
no
|?- functional_divisible(mold1,fabricate_short_shaft_1).
yes

|?- use_specification(mold1,fabricate_short_shaft_1, pd1,
     30,30,recatngle_1).
Q = 30;
no

|?- has_current_activity(mold1, List, tp).
 .
 .
 .
 .
 .  Requires
 .
 .   |?- committed_to(mold1, A, S, _, tp, rectangle_1).
 .   A = fabricate_short_shaft_2
 .   S = es_fabricate_short_shaft_2;
 .   no
 .
 .   |?-enabling_state(fabricate_short_shaft_2,tp,Status).
 .   Status = enabled;
 .   no
List = assemble_clip_reading_lamp, assemble_hand;
no
```

has_current_activity is used to to find out if the activities requiring and supported by the resource are homogenous/ heterogenous

```
|?- total_committed(mold1, TQ, tp, rectangle_1).
TQ = 30;
no

|?- rpl(mold1, Q, tp2, rectangle_1).
Q = 150;
no

|?- simultaneous_use_restriction(fabricate_shirt_shaft_1,
A2, mold1).
no.
```

simultaneous_use_restriction is used so that no two confliction activities can use the resource simultaneously

temporal ontology (e.g before, after)

```
Result = [fabricate_short_shaft_1];
no
```

the resource has the capacity to only support fabricate_short_shaft_1 activity

Assignment of resources (or alternative resources) is a crucial function in a manufacturing environment. This is used in case of resource unavailability, for example, due to a machine breakdown or delay in a shipment.

☞ ☞ ***Can resource*** `mold2` ***be used instead of resource*** `mold1`? (Is mold2 an ***alternative resource*** of mold1 for activity `fabricate_short_shaft`?)

```
|?- alternative_resource(mold1, fabricate_short_shaft, List),
member_of(mold2, List).

      Requires
      |
      |---> activity state ontology.
      |
      |---> |?- role(R2, fabricate_short_shaft_1, tool).
            yes
```

the alternative resources has to have same role

```
List = [mold2];
no
```

mold2 is an alternative resource for the activity fabricate_short_shaft_1

A smooth production requires the availability of resources at the time of productions. Resources should be replenished whenever before the stock reaches a certain limit.

☞ ☞ Is the stock for `short_shaft` in danger of being depleted between now and the end of the week?[1]

```
|?- rp(short_shaft, Q, Tp), Tp > now, Tp < tp333, Q < 20.
Q = 15
Tp = tp23;
Q = 10
Tp = tp200;
no
```

[1] i.e will the resource point fall below then a certain value

Another important function in planning is to be able to explode the bill of material of a resource.

☞☞ *What are the components of resource* `clip_reading_lamp`*?* (What are the ***physical component of*** the resource?)

```
|?- physical_component_of(R, clip_reading_lamp, A, Type).
    Requires
      ├► |?- physical_division_of(R2, clip_reading_lamp).
      └► |?- role(R2, fabricate_short_shaft_1, tool).
             each physical division should not share the
             same role with the original resource

R = clip_base
A = assemble_clip_reading_lamp
Type = physical;
R = short_arm
A = assemble_clip_reading_lamp
Type = physical;
R = small_head
A = assemble_clip_reading_lamp
Type = physical;
no
```

6.2.1 Resource Manger inquiries

This section contains further examples of some queries that are asked by the *resource manager* as described in the Integrated Supply Chain Management project (*ISCM*) developed in the Enterprise Integration Laboratory (*EIL*). The project is being developed for the purpose of evolving the next generation of supply chain integrating functions such as scheduling, production planning, order entry, forecast, inventory management. These functions span needed steps for the fulfilling an order, from the order receival to order shipment. The resource manager answers queries regarding resource availability, loading level, alternative resources and storage related queries. Here some more examples of such queries.

- What are the inactive resources?[1]

[1] i.e which resource has not been used or consumed by any activity in a certain duration

```
|?- \+ committed_to(R, A, _, Ti,_,_), EP(ti) > 20.

A = mold_spacer2;

A = mold_socket_seat_2;

no
```

- Any material shortfall?[1]

```
|?- rp(v_spring, Q, tp1), Q < 0.

no
```

- Which components/parts is assigned to activity ***A*** to produce resource ***R***?[2]

```
|?- consumption_spec(clip_base, assemble_clip_base,
_,_,_), physical_component_of(R2, clip_base), produce(S,
assemble_clip_base), produces(S, clip_base).

R = v_spring

R2 = clip_base

S = produce_state_for_clip_base

A = assemble_clip_base;

R = round_nut

R2 = clip_base

S = produce_state_for_clip_base

A = assemble_clip_base;

R = bolt4

R2 = clip_base

S = produce_state_for_clip_base

A = assemble_clip_base;

R = v_pad

R2 = clip_base

S = produce_state_for_clip_base

A = assemble_clip_base;
```

- What is ***R*** used for?

[1] i.e negative inventory

[2] the resource produced in this case is the *clip base*

```
|?- (use_specification(clip_base, A, _,_,_,_); consump-
tion_spec(clip_base, A, _,_,_,_)), role(clip_base, A,
Role)).
A = assemble_clip_base
Role = raw_material;
no
```

- Is the incoming resource to be sent to storage or to be processed?

  ```
  |?- to_storage(R, Tp, Result).[1]
  Result = to_storage;
  no
  ```

- Where can ***R*** be stored?

  ```
  |?- use_spec(v_spring, store_v_spring, _, _, _),
  use_spec(Area, store_v_spring, _, _, _),
  subclass_of(Area, work_cell).
  Area = loc1;
  Area = Loc4;
  no
  ```

- Can ***R*** be stored in ***S*** over time interval ***ti***?
- Can ***R*** be stored in different location?
- What is the capacity trend of resource ***R***? Is it increasing, decreasing, steady or undetermined?

  ```
  |?- trend(oven, tp9 Result).
  Result = increasing;
  yes
  ```

- If I use ***M*** of ***X*** now, can I also use ***N*** of ***R*** later?

  ```
  |?- available_for(water, [heat_up], pd3,_),
  available_for(water, [heat_up], pd4,_).
  yes
  ```

- What is the load profile of resource ***R*** at time point tp1 and tp3?

  ```
  |?- has_current_activity(oven_1, Act_list1, tp1),
  ```

[1] to_storage(R, TP, Result):-
committed_to(R, A, S, Ti, Q,U),
((before(Tp, Ti), Result = to_storage,!); Result = to_be_processed,!).

```
has_current_activity(oven_1, Act_list2,tp3).
List1 = [bake_small_pizza, bake_large_pizza]
List2 = bake_large_pizza;
no
```

- What is the activity history of resource ***R***?

```
|?- activity_history(assemlby_area_1, Act_list, tp2).
Act_list = [assemble_clip_reading_lamp, assemble_hand];
no
```

- Is the resource ***R*** used/consumed by activity ***A*** on continuous or discrete basis?[1]

```
|?- usage_mode(assembly_area_1, assemble_hand, Type).
Type = discrete;
no
|?- usage_mode(water, drinking, Type).
Type = continuous;
no
```

6.2.2 For future work

This section contains resource related competency questions that are more application dependent; cost accounting, quality are examples of such applications. Such efforts are being developed in TOVE and they include the definition of cost and quality ontology/axioms.

- Is resource ***R*** above loading limits?
- Which resources have exceeded their loading limit?
- From which stock(s) activity ***A*** is covered?
- Who are the suppliers for resource ***R***?
- What is the priority of purchasing of resource ***R***?
- Is there a constraint for moving ***R*** from ***a*** to ***b***?
- What is the reject rate for resource R (part/structure)?
- What is the lead time for arrival time for resource R (part/structure)?-sportation assist for resource ***R***?

[1] Updating the resource point requires the ability to reason about the mode of usage. The term specifies whether the resource is consumed or used on discrete or continuous basis. The usage mode requires the nature of usage and unit of measure terms.

- Which resource/part to replace resource/part ***R*** in structure *S*?
- What is the breakdown specification of resource ***R***?
- Is resource ***R*** out of order?
- Which resource is out of order?
- What is the cause of failure for resource ***R***?
- What is the history of failure of resource ***R***?
- What is the average repair time for resource ***R***?
- What is the expected time for next breakdown of resource ***R***?
- What is the expected time left for resource ***R*** to be phased out? (date of validity)
- When is the next replacing cycle for resource (machine/tool) ***R***?
- What is the reject rate for resource ***R*** (part/structure)?
- What is the lead time for arrival time for resource ***R*** (part/structure)?
- To which order the resource ***R*** belongs to?
- If resource ***R*** is delayed/breaks down/... which other activities that will be affected?
- What is the inventory unit cost rate for part A or structure S?
- What is the cost accounting results for part A or structure S?

6.3 Conclusion

This chapter presented a set of competency questions that should be answered by any manufacturing reference model. Defining enterprise reference models has been the focus of a number of efforts. However these efforts have provided minimal means of evaluating the models. Out of the evaluation criteria defined in chapter one, the competency criterion checks how well a model support problem solving by its ability to answer a set of pre-defined questions. Through the redesigning of the short shaft scenario, a list of competency questions are presented to demonstrate the use of the ontology in inter-disciplinary communication. It also defines the range of tasks supported by the ontology. The chapter also presented some application dependent competency questions that should be addressed by future work.

CHAPTER 7 *Conclusion*

Planning involves selecting and sequencing activities to achieve a goal while scheduling involves assigning resources to activities over a time interval so that they obey temporal constraints and capacity limitations of shared resources. The complexity of planning and scheduling is determined by the degree to which activities contend for resources. These systems have to be able to reason about availability and allocation of shared resources to activities and that requires the ability to reason about the properties of resources when used or consumed by an activity.

In our attempt to define generic ontologies, including a resource ontology, we have faced the following technical problems:

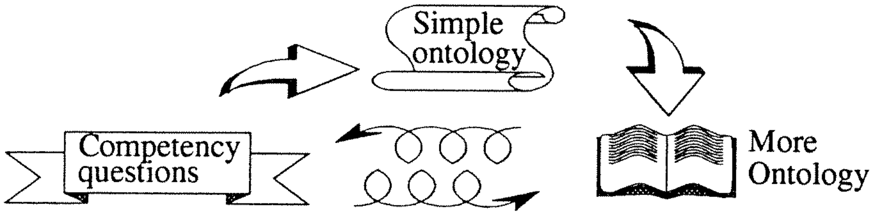
- The first problem that we have faced is that there is no agreed on methodology for ontology definition. There is no crisp set of steps and requirements to be followed. This has led us to develop the methodology, adopted in this thesis, which links ontology development with the competency questions. Moreover not much research has focussed on how to evaluate and validate such efforts.
- We are striving towards defining a "generic" resource ontology to be reusable by different applications in "any enterprise"; however this has proven to be difficult and problematic. The scope is too abstract and unbounded resulting in the definition of numerous terms in the ontology with no definite vision of their usefulness. Based on this, the focus of formalization has changed from a general enterprise to a manufacturing enterprise. The goal has become the definition of a generic

resource ontology applicable in a manufacturing environment. In other words, we must define an ontology to be used by planning and scheduling activities to reason about shared resources.

- With the focus changing to manufacturing, the question becomes what to represent. What is the minimal and most critical information required to represent a resource? How is this information going to be represented? Are they going to be represented as assertions or as axioms? Addressing the issue of what to represent we have integrated terms available in existing modelling, planning and scheduling efforts with new ideas. In this way, we ensure that the resource ontology will be able to represent problems addressed by other efforts.

In our methodology, we have chosen the competency requirement as the focal point of the effort. Competency questions represent the starting point of the ontology development, as well as defining the types of tasks that the representation can be used in. The driving element in the ontology creation is the definition of a set of stratified questions, which in turn initiates the process of defining a simple and primitive ontologies. Based on such ontologies, more complex terms are defined. Hence, the process becomes a continuous iterative process between the ontology and competency modifications [figure 60].

FIGURE 60 The used methodology



As mentioned, evaluation and validation of such efforts is problematic. Through the development of the resource ontology, we think we have fulfilled these requirements:

- **generality** is satisfied as the ontology is used by a number of different ontological efforts: planning, scheduling, cost accountancy[1], quality[2] and purchasing[3] ontology.

[1] Currently being developed in TOVE environment [Tham & Fox 94]

[2] An on going endeavor within TOVE is to create quality ontology [Kim & Fox 94]

[3] Nothing has been applied in this domain through the laboratory however there is an on-going research at IBM to develop purchasing ontology [Grosof 92].

- **competency** is satisfied through the definition of questions that has to addressed by the model.
- **granularity** is satisfied through the implementation of the definitions and constraint in Prolog.
- **perspicuity** is satisfied with the availability of the descriptive methodology.

The resource ontology, together with the activity-state, time and causality ontologies constitute the basis or the core ontology for on-going projects in the Enterprise Integration Laboratory; efforts such as: formalization of quality, cost, time-based competition ontology. Finally, it has furnished similar efforts with the methodology that is described before.

I have presented a synopsis of my accomplishments in my attempt to present a generic resource ontology that are sharable and reusable across various enterprise domains ranging from cost accounting to scheduling. In the next section I will be presenting recommendations for future related work.

Recommendation for future work:

- **Continuous appraisal:**
  Continuous appraisal of the ontology as a result of the development of application dependent ones is a must. (ie the inclusion of application dependent ontology in the generic model if these ontologies are shared by a number of activities)
- **Genericity of the model:**
  The issue that is raised is concerning how generic the ontology is? The test of genericity is determined only by the consistent use of the ontology in a variety of applications. This is achieved through the use of the Integrated Supply Chain Management environment (ISCM). "The supply chain is a set of activities that span enterprise functions from the ordering and receipt of raw materials through the manufacturing of products through the distribution and delivery to the customer" [Fox et al 93b]. The goal of this project is to create an integrated environment between different enterprise functions in order to operate efficiently and effectively and to increase the market response time. Different agents require different information regarding resources. The "scheduling agent" requires information about resource availability; "design

agent" requires information on design dimension; "resource agent" requires information concerning bills of material explosion information.

Also the attention should be directed towards the Design In the Large (DIL) project to represent its requirements. Although number of the described terms are applicable to DIL environment, but not much work has been done in this area. One of the issues addressed in the "DIL" context is detailed part description for example; this could have information that might be included in the generic resource ontology.

- **Capacity recognition process**:
  The process includes a sequencing heuristic for one machine that supports multiple activities/jobs. The heuristic is used for independent activities. Future work related to the heuristic should include the capacity heuristic in a temporal constraint propagation algorithm. Initial research on the subject is presented in appendix B.
- **Aggregation of resources**:
  The notion of aggregation/grouping of resources, on the basis of over lapping capacity for example, is not considered in this endeavour. This is because grouping in such manner is application dependent and should be defined by different applications (agents). However, again if it turns out that there exist a number of resource aggregation(s) that is generic, then they should be added. The same is true for hierarchial classification of resources. The "logistics agent" might define an "industrial plant" as consisting of a group of "work areas" which in turn consist of "work cells"; a "work cell" has "machines" and "tools" as constituents.
- **Evaluation of the model with respect to the rest of the criteria**:
  The resource ontology was not tested against: transformability, scalability, extensibility, and efficiency criteria. Therefore future work should be directed towards testing these criteria. Transformability, scalability and extensibility criteria can only be evaluated through the consistent use of the representation in a variety of applications. The efficiency criterion is problematic as it has been shown that there is more than one way to represent the same knowledge and each way does

not have the same complexity to answer queries. This is an area which requires the focus of more future work.

APPENDIX A Ontology in other models

This appendix compares the three reference models: IWI, CIM-OSA and PERA, with respect with the evaluation criteria defined in chapter one. Moreover the reviewed literature, are checked whether they contain the defined ontology in this thesis, or whether they contain similar notions.

A.1 Ontology in the other efforts

TABLE A.1 Resource ontology in other efforts

Legend:

✗ Not available
✓ Available
☞ See table footnote
✎ Could be available if added to the model
● Difficult to achieve
✉ Could not be checked

Ontology	IWI	CIM-OSA[a]	PERA[b]	SIPE	CYC	KRSL	OPIS	Gerry
Resource known	✓	✗	✗	✓	✓	✗	✗	✗
Taxonomic classification	✓	✓	✓	✓	✓	✓ ☞[c]	✓	✓
Role	✗	✗	✗	✗	✗	✗	✗	✗
Physical Division of	✓	✓	✓	✓	✓	✓	✓	✓
Functional Division of	✗	✗	✗	✗	✗	✗	✗	✗
Temporal Division of	✗	✗	✗	✗	✗	✗	✗	✗

TABLE A.1 Resource ontology in other efforts

Legend:

✗ Not available

✓ Available

☞ See table footnote

✎ Could be available if added to the model

● Difficult to achieve

✉ Could not be checked

Ontology	IWI	CIM-OSA[a]	PERA[b]	SIPE	CYC	KRSL	OPIS	Gerry
Mobile resource	✗	✗	✗	✗	✗	✓ ☞[d]	✓	✗
Stationary resource	✗	✗	✗	✗	✗	✗ ☞[e]	✓	✗
Alternative resource	✓	✓	✗	✗	✗	✗	✓	✓
Temporal divisible	✗	✗	✗	✗	✗	✗	✗	✗
Physical divisible	✗	✗	✗	✗	✗	✗ ☞[f]	✗	✗
Functional divisible	✗	✗	✗	✗	✗	✗	✓	✗
Physical component of	✓ ☞[g]	✓ ☞[h]	✓	✓	✓	✓	✓	✓
Functional component of	✗	✗	✗	✗	✗	✗	✓	✗
Unit of measurement	✓	✓ ☞[i]	✓	✓	✓	✓ ☞[j]	✓	✓
Measured by	✗	✓	✗ ☞[k]	✗	✗	✓	✓	✓
Nature of usage	✗	✗	✗	✗	✗	✗	✗	✗
Resource point	✓ ☞[l]	✓	✗ ☞[m]	✓ ☞[n]	✓	✓ ☞[o]	✓	✓ ☞[p]
Resource point encapsulation	✗	✓	✗ ☞[q]	✗	✗	✓ ☞[r]	✓	✓
Usage mode	✗	✗	✗	✓	✗	✗	✗	✗ ☞[s]
Resource exist	✗	✗	✗	✗	✓	✗	✗	✗
Consumption specification	✓ ☞[t]	✓ ☞[u]	✗ ☞[v]	✓ ☞[w]	✗	✓ ☞[x]	✓	✓
Use specification	✓ ☞[y]	✓ ☞[z]	✗ ☞[aa]	✓ ☞[ab]	✗	✓ ☞[ac]	✓	✓

TABLE A.1 Resource ontology in other efforts

Legend:

✘ Not available
✔ Available
☞ See table footnote
✎ Could be available if added to the model
● Difficult to achieve
✉ Could not be checked

Ontology	IWI	CIM-OSA[a]	PERA[b]	SIPE	CYC	KRSL	OPIS	Gerry
Production specification	✔	✔	✘	✔	✘	✔	✔	✔
Release specification	✘	✘	✘	✘	✘	✘	✘	✘
Resource configuration	✔	✘	✘	✘	✘	✘	✔	✔
Simultaneous use restriction	✘	✉	✘	✔ ☞ ad	✘	✔ ☞ae	✘	✔
Committed to	✔	✉	✘	✔ ☞ af	✘	☞ag	✔	✔
Total committed	✘ ✎	✉	✘	✔	✘	✘ ✎	✔	✔
Has current activity	✘ ☞ ah	✘ ✎	✘	✎ ☞ ai	✘	✔ ☞aj	✘	✔
Available for activities	✘	✘	✘	✘	✘	✔ ☞ak	✘	✘
Available capacity	✘	✘	✘	✘	✘	✔ ☞al	✔	✘
Trend recognition	✘	✘	✘	✘	✘	✘	✘	✘
Activity history	✘ ☞ am	✘ ✎	✘	✔	✘	✘	✘	✔

[a] Although a detailed resource view is not included in any of the references, similar ontology are abstracted from constructs in the function and information view. The resource behind not including the resource view in any of the AMICE documents is because up to date the resource view and it constructs have not yet been developed [CIM-OSA 91a]

b. Purdue [PERA 91] & [IFAC/IFIP 93] does not contain neither a resource modelling methodology nor a resource model. Purdue's architecture so far describes the life cycle of **creating a reference architecture**. The model is descriptive. Based on that most of the compared ontology are not included in the model.

c. Through using "is-a" relation.

d. An object is mobile if it is defined as being "Transport Capacity".

e. An object is stationary if it is not defined as being "Transport Capacity".

f. Not explicitly defined but it is implied if the object is defined as being "sharable".

g. Through the part and structure relationship.

h. Through the use of *comprises* slot in the *object class* construct defined in the function view. Furthermore, the same information could abstracted using the *related objects* slot the *enterprise object view* construct defined in the information view.

i. Defined as slot in the *required capabilities* constructs.

j. Defined as "unit-type" & "measured-by" slots in the resource frame definition.

k. Although it is an essential and basic concept in any model but since none of the PERA's document contain any information on the issue, it is mentioned that the model lacks the term.

l. Through the "Inventory level data" slot in the "PART" entity.

m. Same as footnoote k.

n. Through using "level" predicate.

o. Defined as "quantity" slot in the resource frame definition.

p. Amount of a resource is calculated from the resource's activity history.

q. Same as footnoote k.

r. through "unit" relation.

s. Only discrete applications are represented.

t. The consumption specification is implied through accessing "quantity" slot on "PRODUCTION ORDER" entity.

u. Through the use of "required capabilities" construct.

v. Same as footnoote k.

w. Defined in the "Put on" operator

x. Achieved through accessing "allocation-event" or "de-allocation-events" or "production-events" attributes in the resource frame definition.

y. The use specification is implied through accessing:

- "PERSONNEL ASSINGMENT" relation between "PERSONNEL" and "EQUIPMENT GROUP" entities,
- "OPERATION ASSIGNMENT" relation between "OPERATION" and "EQUIPMENT GROUP" entities
- "TOOL ASSIGNMENT" relation between "OPERATION ASSIGNMENT" and "TOOL USE".

z. Through the use of "required capabilities" construct.

aa. Same as footnoote k.

ab. Same as footnote w.

ac. Same as footnoote x.

ad. A resource, by definition, can not be shared by two activities

ae. Same as footnoote x.

af. Implied when a resource is declared as a resource with respect to an activity.

ag. Defined through "allocate-consumable-resource" event. This event would check if the resource is available or not and commit the resource. It was not mentioned how this would apply on reusable resources.

ah. Not explicitly defined, however this information is acquired through accessing "MACHINE LOADING" relation.

ai. Through checking which activity the resource is linked to.

aj. Done through accessing "Resource usage Records" which provides meta information about which activities (events) that are supported by the resource.

ak. KRSL does not contain capacity assessment theory. If a resource is defined as being sharable then the resource can support more than one event (activity). This is explicitly stated using "is-a" relation that defines an object as being "sharable". However a resource could be checked whether it is available or not through accessing the value of "quantity" slot in the resource frame.

al. Done through accessing "Resource usage Records" through accessing the value of "Available Capacity" slot in the resource frame.

am. Same as footnoote ah.

APPENDIX B *Capacitated temporal constraint propagation*

This appendix presents initial research on defining a capacitated temporal constraint propagation.

B.1 Introduction

In chapter five, a methodology for defining a resource's capacity is defined. A sequencing heuristic is presented that sequences "independent" jobs on a single machine that can support multiple activities simultaneously.

This appendix presents initial research for including the capacitated sequencing heuristic. This is done through the use of a heuristic due to [Chu & Ngai 93] for embedding temporal constraint propagation in machine sequencing for job shop scheduling. Unlike the sequencing heuristic, defined in chapter four, the new heuristic (*Capacitated Temporal Constraint Propagation for Machine Sequencing*[1]) sequences activities on a resource(s) taking into consideration the precedence constraint between activities inaddition to resource constraints.

B.2 About the heuristic

The heuristic presented in figure B.1 and figure B.2 defines how activities are chosen to be scheduled. Moreover, based on the selection of an activity, the capacity and temporal constraints are propagated.

[1] CTCP

The **CTCP** heuristics is as follows is as follows:

Phase 1:

1. Identify the starting time window for all activities.
2. Let **[S]** be a list of the activities that have no preceding activities and the activities with all preceding activities are scheduled.
3. Choose activity **X** from **[S]** with the EDD.
4. Find all activities with earliest starting time less or equal to the earliest finish time of activity **X** (i.e $St_e(Y) \le Et_e(X)$). Put the activities in **[C]**.
5. Select an unmarked (i.e unscheduled) activity **Z** from **[C]**, and assign to it the earliest starting time ($St_e(Z)$).
6. Propagate the capacity constraint as a result of activity **Z** using a resource over the whole duration of the activity.
7. If the capacity constraints are unsatisfied[1], then mark the activity **Z** as and **go to step 5** else continue.
8. Propagate temporal constraints.
9. If an empty window start is formed during the temporal constraint propagation, then the current sequence of activity **Z** is infeasible. Mark the activity and **go to step 5** else, if the sequence is still feasible, then continue.
10. If all activities are scheduled, then **go to phase 2 (step 11)**, else **go to step 2.**

Phase 2:

11. If there are *marked activities*, then schedule them after the last scheduled activity in phase 1, else continue.
12. End of heuristic.

[1] i.e infeasible sequence.

FIGURE B.1 CTCP - phase 1

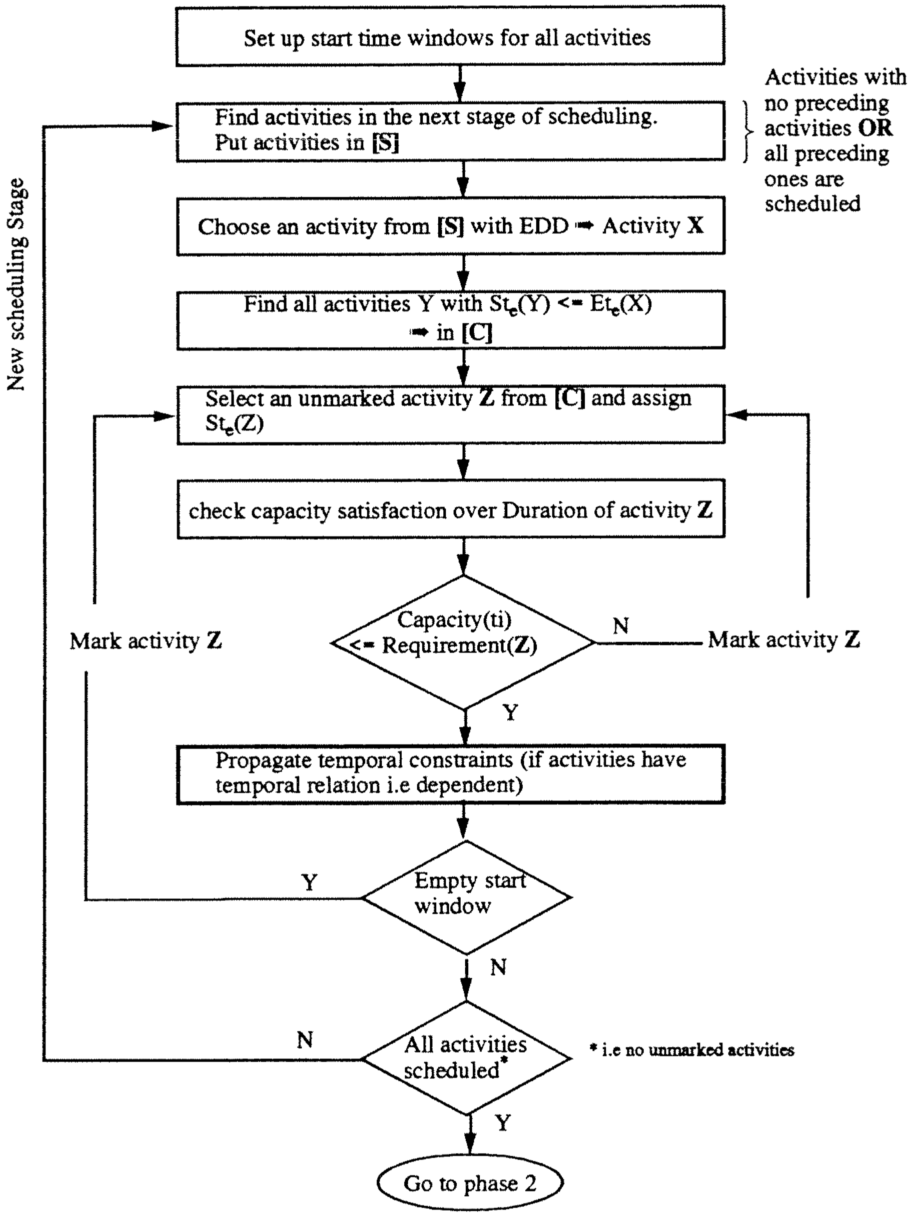
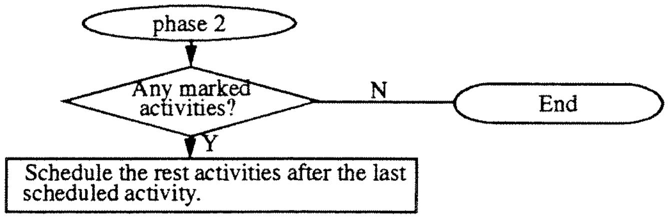

Set up start time windows for all activities
Find activities in the next stage of scheduling. Put activities in [S]
Activities with no preceding activities **OR** all preceding ones are scheduled
New scheduling Stage
Choose an activity from [S] with EDD ➟ Activity **X**
Find all activities Y with $St_e(Y)$ <= $Et_e(X)$ ➟ in [C]
Select an unmarked activity **Z** from [C] and assign $St_e(Z)$
check capacity satisfaction over Duration of activity **Z**
Mark activity **Z**
Capacity(ti) <= Requirement(**Z**)
N
Mark activity **Z**
Y
Propagate temporal constraints (if activities have temporal relation i.e dependent)
Y
Empty start window
N
N
All activities scheduled*
* i.e no unmarked activities
Y
Go to phase 2

FIGURE B.2 CTCP - phase 2


phase 2
Any marked activities?
N
End
Y
Schedule the rest activities after the last scheduled activity.

APPENDIX C *Code listing*

This appendix contains the prolog and C code implemented in the thesis. The prolog code contains the axioms and the ground terms defined in the ontology chapter. The C code is the programming environment inwhich the sequencing heuristic is implemented in.

C.1 Sequencing heuristic

```
/*---------------------------------------------
    Module:     Capacitated Sequencing heuristic {see chapter four}
    Author:     Fadi George Fadel
    Purpose:    Sequences independent jobs on a single resource that has the capa-
                bility to support multiple jobs simultaneously
    Comment:    Four detailed information of the heuristic used, check chapter four.
                This version is the ESR one. Since difference between this version
                and the rest is in the assignment criteria, therefore it is going to be
                pointed out in the code what has to be changed in each version.
/*---------------------------------------------*/
#include <stdio.h>
#define begin {
#define end }
#define True 1
#define False 0
#define Max_N 160 /* Number of jobs */
#define Max_t 200 /* time span of N resource */
int i,j,k,N,t,
    Sequence_Is_Done;   /* Flag for the completion of te sequence */

struct job {
    int job_no;          /* JOb ID */
    int r_job;           /* Ready Time */
    int d_job;           /* Due Date */
    int p_job;           /* Processing Time */
    int c_job;           /* Capacity Requirements */
    int status;          /* 0 if not scheduled & 1 if scheduled */};
struct job all_job[Max_N]; /* Array of structure of jobs */

struct scheduled_job {
    int job_no;
    int st_job;
    int et_job;
    int status;
    int tm_period;};
struct scheduled_job all_scheduled_job[Max_N]; /* Array of structure of jobs */

struct EDD {
    int job_no;
    int EDD;
    int st;};
struct EDD EDD_sequence[Max_N];

int Conflict_job[Max_N];

struct Resource_Capacity {
    int amount;};
struct Resource_Capacity Current_Resource_Capacity[Max_t];

/*------------------- Main-------------------------*/
main (argc, argv )
    int argc;
    char *argv[];
begin /* Main */
/*------------------- Initailize-------------------------*/
  for (k = 0; k < N;k++)
  { EDD_sequence[k].job_no = k+1;
  EDD_sequence[k].EDD = 0;}
  for (k = 0; k < t;k++)     /* Initialize Current_Resource_Capacity */
  {
  Current_Resource_Capacity[k].amount = 150;}
  for(k = 0; k<N; k++) {
       all_scheduled_job[k].status = 0;
       all_job[k].status =0;}
  for(i = 0; i<N; i++) Conflict_job[i] = 0;
  Sequence_Is_Done = False;
  /*--------------------- Initailaze--------------------- */
  Read_Input (argc, argv);                    /* Calling Read_data function */
  EDD_Stack(EDD_sequence, N); /* Sequence the jobs according to EDD */

while(!Sequence_Is_Done) /* Try to sequence all jobs to minimze the number of */
                              /* tardy jobs */
{
  Schedule_w_EDD();      /* Schedule according to the EDD */
  Check_Capacity();      /*Find the capacity used at all time periods */
  Capacity_Conflict();   /* Check violation of capacity */
}

CR_Disregarded_jobs_2(); /* Schedule the disregatded jobs according to CR after */
                              /* the first phase of sequencingis done.
                              - this function:
                               : sequence jobs with 0<= CR <= 1 after the milestone
                              : sequence jobs with CR<=0 and jobs not scheduled from
                              0<= CR<=1 by checking if there is any available time slot
                              that can not violate the capacity constraint */
end /* Main */
/* ----------------ReadInput - Function -------------------------- */
Read_Input( argc, argv )
/* Phase1:
  This function reads the input file. The first line of the data file should contain the
  number of jobs then the length of the time window considered. The rest of the lines
  should contain the job number, ready time, due date, processing time and capacity
  requirement of each job */

  int argc;
  char *argv[];

begin /* Read_Input */
  int i,j,k;
  FILE *fp, *fopen(), *in_file;
  if (argc == 1) {
```

```
  printf("Usage: CCTSP datafile.\n");
  exit(1);}

if( (in_file = fopen(*++argv, "r")) == NULL) {
  printf("Can't open %s\n", *argv);
  fclose(fp);
  exit(1);}
/* Reading data */
fscanf(in_file,"%d %d",&N,&t);
for (i = 0; i < N; i++) {
  fscanf(in_file,"%d %d %d %d %d", &all_job[i].job_no, &all_job[i].r_job,
                        &all_job[i].d_job,
                        &all_job[i].p_job,
                        &all_job[i].c_job,
                        &all_job[i].status);} /* for i */
fclose(in_f3ile);
end /* Read_Input */
/* -------------Read_Input -End --------------------------*/
/* -------------EDD_Stack -function-----------------------*/
 /* Phase 1 cont'd:
   This function sorts jobs according to EDD rule */

EDD_Stack()
begin
  int i,j,k,temp ;
  int swapped;

/* getting all due dates from all_job[x].* to be saved in EDD_sequence[x].* */
 for (j=0; j<N; j++)
{EDD_sequence[j].job_no = all_job[j].job_no;
EDD_sequence[j].EDD = all_job[j].d_job;}

/* Sort the jobs according in ascending order of the EDD */
 i = 0;
swapped = True;
while ((i < (N-1)) & swapped){
  j = N-1;
  swapped = False;
  while (j > i){
      if (EDD_sequence[j].EDD < EDD_sequence[j-1].EDD)
```

```
        {
        swapped = True;
        temp = EDD_sequence[j].EDD;
        EDD_sequence[j].EDD = EDD_sequence[j-1].EDD;
        EDD_sequence[j-1].EDD = temp;
        temp = EDD_sequence[j].job_no;
        EDD_sequence[j].job_no= EDD_sequence[j-1].job_no;
        EDD_sequence[j-1].job_no= temp;
        } /* for */
  j = j -1;
  } /* while j */
  i = i +1;
  } /* while i */
end                            /* EDD_Stack */
/*------------EDD_Stack- End----------------------------*/
/*------------Schedule_w_EDD - function-----------------*/
/* Phase 1 cont'd:
  This function just assigns jobs to time slots in the resource horizon without check-
  ing capacity */

Schedule_w_EDD()
/* Scheduling of jobs according to EDD without checking Capacity */
begin
  int i,j,k;
  int go_to,temp1,temp2,temp3;
  char chr;


for(i=0; i<N; i++) {
/* 1. Go through EDD_sequence array & schedule each job in its time window.
  Note: the status should be checked so that for the capacity recheck, all the deleted
  jobs are excluded.
  2. calculate et of the job & save it in all_scheduled_job[i].et_job*/
/* 1, 2*/
/* Find job index */
temp1 = EDD_sequence[i].job_no;
go_to = temp1 - 1;
/* After the first phase jobs that disregarded should be removed from the sequence *
if(all_job[go_to].status == 1)              /* go_to -> index */
{all_scheduled_job[i].job_no = 0;
all_scheduled_job[i].st_job = 0;
```

```
all_scheduled_job[i].et_job = 0;}

if(all_job[go_to].status == 0) {
  all_scheduled_job[i].job_no = EDD_sequence[i].job_no;
  all_scheduled_job[i].st_job = all_job[go_to].r_job;
  all_scheduled_job[i].et_job = all_scheduled_job[i].st_job + all_job[go_to].p_job;
  } /* if(all_job[go_to].status == 0)*/

  /*
    For ESSR & LSPR versions the above "if" statement is going to be replaced y
    the following:
      if(all_job[go_to].status == 0) {
          all_scheduled_job[i].job_no = EDD_sequence[i].job_no;
          all_scheduled_job[i].st_job = all_job[go_to].d_job;
          all_scheduled_job[i].et_job = all_scheduled_job[i].et_job -
                                                 all_job[go_to].p_job;
      } /* if(all_job[go_to].status == 0)*/
  */

} /* for i */
end                          /* Schedule_w_EDD */
/*------------Schedule_w_EDD - End------------------*/
/* ----------------Check_Capacity -function------------------------ */
/* Phase 1 cont'd:
  This function checks for capacity violation of the resource over a specified resource
  time window */

Check_Capacity()
begin
  int time_now,sequence_no,i, j;
  int go_to,temp1,temp2,temp3,temp4;

  sequence_no = 0;                         /* Intializing */
/* Assigning the jobs to time periods instead of tm points in all_scheduled_job */
  for(i = 0; i < N; i++)
  { all_scheduled_job[i].tm_period = all_scheduled_job[i].st_job;}

  /* Capacity check for time periods */
  for(i = 0; i<N; i++){
      temp1 = EDD_sequence[i].job_no;
      go_to = temp1 - 1;  /* index is based on the EDD_sequence[i].job_no *
      if(all_job[go_to].status == 0)           /* go_to -> i */
      {
          temp2 = all_scheduled_job[i].tm_period; /* starting period of the job */
          temp3 = (all_job[go_to].p_job + temp2) - 1; /* Ending period of the job */
          for(j = temp2; j < (temp3+1); j++){
          /* Update the Resource Capacity over the processing time period of job */
          Current_Resource_Capacity[j].amount =
           Current_Resource_Capacity[j].amount - all_job[go_to].c_job;
          } /* for j */
      } /* if(all_job[i].status ==0) */
  } /* for i */
end /* Check_Capacity*/
/* ----------------Check_Capacity -End----------------------- */
/* ----------------Capacity_Conflict -function------------------------ */
/* This function finds the period(s) inwhich capacity is violated */

Capacity_Conflict()
begin
  int i,j,k,c,temp1,temp,go_to;
  int swapped,
  no_violation,
  req,
  p_req,
  for_job;

  no_violation = True;
  req = 0; /
  /* variable used to find the job with the highest capacity requirement*/
   p_req = 0;                 /* varaiable to find the highest procsessing time*/
  for_job = 0 ; temp = 0; i = 0;

  while((no_violation) && (i<(t+1))) /* while capacity is not violated */
  {if(Current_Resource_Capacity[i].amount < 0){
      /* checkimg if the capacity is violated */
      no_violation = False;
      for(j=0; j<N; j++) /* find the jobs using the resource at the time of violation */
      { /* Find the job scheduled at time of violation */
      if((all_scheduled_job[j].tm_period <= i) &&
                        (all_scheduled_job[j].et_job >= (i+1)) )
      {
          go_to = (all_scheduled_job[j].job_no) - 1;
          /* index of the job to use in all job structure */
          /* Finding: - the job with the highest usage requirement in terms or
           number of blocks OR
          - the job with highest requirement AND longest pj */
          if (((all_job[go_to].c_job >=req ) &&
                      (all_job[go_to].p_job >p_req)) ||
                      (all_job[go_to].c_job > req))
          /*
            For the LSR & LSPR only above “if” condition is going to be replaced
            by:
             if((all_job[go_to].p_job >p_req))
          */
          {
              req = all_job[go_to].c_job;
              p_req = all_job[go_to].p_job;
              for_job = all_scheduled_job[j].job_no;
          } /* if(all_job[go_to].c_job > req) */
      } /* if(all_scheduled_job[j].tm_period <= i)*/
      } /* for j */
      all_job[for_job-1].status = 1;
          /* Disregard the job from the sequence i.e change the status of the job */
  } /* if(Current_Resource_Capacity[i].amount < 0) */
  i = i +1;
  } /* while */

if(no_violation){
  Sequence_Is_Done = True;
  printf(“\n*****> The Sequence is Done\n”);
  printf(“\n all_scheduled_job[i] -> Schedule_w_EDD \n”);
  printf(“ -------------------\n”);
  for(i = 0;i<N; i++)
  printf(“job = %d st_job = %d et_job = %d status= %d\n”,
      all_scheduled_job[i]job_no, all_scheduled_job[i].st_job,
      all_scheduled_job[i].et_job, all_job[(all_scheduled_job[i].job_no)-1].status);
printf(“\n Current_Resource_Capacity[i] ->\n”);
printf(“ ------------------------------\n”);
for(i = 0;i<t; i++)
  printf(“period = %d Current_Resource_Capacity = %d\n”,i,
        Current_Resource_Capacity[i]);
} /* no_violation*/

if(!Sequence_Is_Done){
  for (k = 0; k < t;k++)
  /* Initialize Current_Resource_Capacity so that next phase could use the same
  structure*/
  { Current_Resource_Capacity[k].amount = 150;}
  } /* !Sequence_Is_Done */
end /* Capacity_Conflict */
/* ----------------Capacity_Conflict -End------------------------ */
/* ----------------CR_Disregarded_jobs_2 - Start----------------- */
/* Phase 2 & 3 & 4:
  This function schedule the disregarded jobs, from phase 1, according to "C"ritical
  "R"atio after the first phase of sequencing is done.
  - this function:
        : sequence jobs with 0<= CR <= 1 after the milestone
        : sequence jobs with CR<=0 and any unscheduled jobs with 0<= CR<=1 */

CR_Disregarded_jobs_2()
begin
  int i, j, k, L, T, go_to, milestone, loc_in_EDD, o, swapped, temp, st, et, st2,
  st_counter, END, reset_st, satisfied, not_at_this_tp, tardy, tardy_counter,
  unsequenced_counter, done, available_period, slack, dummy_st_counter, LS;

  float homar;

  struct p_cr {
        int job_no;
        float JCR;
        int status;};
  struct p_cr p_cr_job[Max_N];
            /* Contains jobs with (+)ve & <= 1 CR at the milestone */

  int p_cr_no,              /* p_cr_no-1:index to the no of jobs in p_cr_job[N] */
  o_cr_no;                  /* o_cr_no-1:index to the no of jobs in o_cr_job[N] */
  float CR;

  satisfied = True;
  not_at_this_tp = False;
  /* Initialize p_cr_job & o_cr_job */
  for(i=0; i<N; i++)
  { p_cr_job[N].job_no = o_cr_job[N].job_no = 0;
  p_cr_job[N].JCR = o_cr_job[N].JCR = 10000;
  p_cr_job[N].status = o_cr_job[N].status = 1; /* i.e a disregarded job */
  } /* for i */

  p_cr_no = o_cr_no = 0;
  /* 1.. Get the end time of the last scheduled job in phase one (milestone) */
  milestone = -1;
  for(j=0; j<N; j++)
  {
  if(all_scheduled_job[j].et_job > milestone)
      milestone = all_scheduled_job[j].et_job; }

/* 2.. Find disregarded jobs ie the ones with all_job[].status == 1
  3.. Save in: a.. p_cr_job if CR is (+)ve & <= 1
  4.. o_cr_job if CR is (-)ve & > 1 */

  p_cr_no = o_cr_no = 0;
  CR = 0.00000;
  for(j=0; j<N; j++)
  {
  go_to = EDD_sequence[j].job_no - 1;
  /* 2.. */
  if(all_job[go_to].status == 1)
  {
      if(milestone == all_job[go_to].d_job)
      { /* if true then CR is infinity then save in o_cr_job[o_cr_no] */
          o_cr_job[o_cr_no].job_no = all_job[go_to].job_no;
          o_cr_job[o_cr_no].JCR = -10000;
          o_cr_job[o_cr_no].status = 1;
          o_cr_no = o_cr_no + 1;
      } /* if milestone */
      else /* if milestone \= all_job[go_to].d_job */
      {
      CR = (float) (all_job[go_to].p_job)/(all_job[go_to].d_job - milestone);
      if((CR > 0.00000) && (CR <= 1.00000))
          {p_cr_job[p_cr_no].job_no = all_job[go_to].job_no;
          p_cr_job[p_cr_no].JCR = CR;
          p_cr_job[p_cr_no].status = 1;
          p_cr_no++;
          } /* if CR > 0 */
          else
          if((CR > 1.00) || (CR < 0.00))
          {
          o_cr_job[o_cr_no].job_no = all_job[go_to].job_no;
          o_cr_job[o_cr_no].JCR = CR;
          o_cr_job[o_cr_no].status = 1;
          o_cr_no++;
          } /* if CR > 1 */
      } /* if milesyone \= all_job[go_to].d_job */
  } /* all_job[got_to].status == 1 */
  } /* for j */

/* 4.. Sort the jobs, in p_cr_job, according in descending order of the good CR*/
temp=0; homar=0.0; i = 0; swapped = True;
while ((i < (p_cr_no-1)) & swapped)
{ j = p_cr_no-1;
  swapped = False;
  while (j > i){
      if ( ((p_cr_job[j].JCR > p_cr_job[j-1].JCR)&&
              (all_job[p_cr_job[j].job_no-1].p_job))||
              (p_cr_job[j].JCR > p_cr_job[j-1].JCR))
      {
          swapped = True;
      homar = p_cr_job[j].JCR;
      p_cr_job[j].JCR = p_cr_job[j-1].JCR;
      p_cr_job[j-1].JCR = homar;
      temp = p_cr_job[j].job_no;
      p_cr_job[j].job_no= p_cr_job[j-1].job_no;
      p_cr_job[j-1].job_no= temp;
      temp = p_cr_job[j].status;
      p_cr_job[j].status= p_cr_job[j-1].status;
      p_cr_job[j-1].status= temp;
 } /*if */
 j = j - 1;
 } /* while j */
 i = i + 1;
 } /* while i */
```

```
/* 5.. schedule the jobs with 0<= CR <= 1 starting from the milestone */
for(j=0; j<p_cr_no; j++)
{go_to = p_cr_job[j].job_no - 1;
 satisfied = True;
 not_at_this_tp = False;

 /* 6.. scheduling disregarded jobs starting from the last ET in the sequence
 (milestone)*/
 for(T=milestone; T<(t+1);T++)
 {satisfied = True;
 while((satisfied) && (!not_at_this_tp) &&
         ((T+all_job[go_to].p_job)<=all_job[go_to].d_job))
 /* 7.. Continue as long as the resource can still hold the job */
 { for(k=T; k<(T+all_job[go_to].p_job); k++)
 /* 8.. checking if the resource can hold the resource for the whole job
     processing time */
 {if(Current_Resource_Capacity[T].amount < all_job[go_to].c_job)
     { satisfied = False;}
 if(k > all_job[go_to].d_job)
     {satisfied = False;}
 } /* for k*/
 if(satisfied) not_at_this_tp = True;
 if(satisfied)
 /* 9.. scheduling the job */
 {for(o=0; o<N; o++)
     if(EDD_sequence[o].job_no == all_job[go_to].job_no)
         loc_in_EDD = o;
     all_scheduled_job[loc_in_EDD].job_no = all_job[go_to].job_no;
     all_scheduled_job[loc_in_EDD].st_job = k - all_job[go_to].p_job;
     all_scheduled_job[loc_in_EDD].et_job = k;
     all_job[all_scheduled_job[loc_in_EDD].job_no-1].status = 2;
     p_cr_job[j].status = 2;

     /* 10.. Update Capacity */
     for(L=T; L < (T+all_job[go_to].p_job); L++)
         { Current_Resource_Capacity[L].amount =
             Current_Resource_Capacity[L].amount all_job[go_to].c_job;}
     }/* if satisfied */
 }/* while */
```

```
  }/* for T */
}/*for j*/

/* 11.. append the job from p_cr_job, that has not been scheduled, to the o_cr_job[].
i.e check all_job[] == 1 */
for(i=0; i<p_cr_no; i++){
  if (p_cr_job[i].status == 1){
      o_cr_job[o_cr_no].job_no = p_cr_job[i].job_no;
      o_cr_job[o_cr_no].JCR = p_cr_job[i].JCR;
      o_cr_job[o_cr_no].status = 1; /* indicating being scheduled in the 3rd phase */
      o_cr_no++;
  } /* if */
} /* for i */

/* 12.. sort o_cr_job according to ascending order*/
i = 0; swapped = True;
while ((i < (o_cr_no-1)) & swapped){
  j = o_cr_no-1;
  swapped = False;
  while (j > i){
      if (o_cr_job[j].JCR > o_cr_job[j-1].JCR)
      {   swapped = True;
          temp = o_cr_job[j].JCR;
          o_cr_job[j].JCR = o_cr_job[j-1].JCR;
          o_cr_job[j-1].JCR = temp;
          temp = o_cr_job[j].job_no;
          o_cr_job[j].job_no= o_cr_job[j-1].job_no;
          o_cr_job[j-1].job_no= temp;
          temp = o_cr_job[j].status;
          o_cr_job[j].status= o_cr_job[j-1].status;
          o_cr_job[j-1].status= temp;
      } /* if */
      j = j -1;
      } /* while j */
      i = i +1;
      } /* while i */

/* 13.. Phase 3:
      try to schedule jobs in o_cr_job from st to the et of the job */
for(j=0; j<o_cr_no;j++){
```

```
go_to = o_cr_job[j].job_no -1;
satisfied = True;
st = all_job[go_to].r_job;
et = all_job[go_to].d_job;
available_period = all_job[go_to].d_job - all_job[go_to].r_job;
slack = available_period - all_job[go_to].p_job;
dummy_st_counter = slack + 1;
LS = all_job[go_to].r_job + dummy_st_counter;
T=st;
done = False;
while((!done) && (T<LS)){
    st2 = T+all_job[go_to].p_job;
    i=T;
    st_counter = T;
    /* 14.. check if the resource can support the job */
    satisfied = True;
    while((satisfied) && (i<st2)){
        if(Current_Resource_Capacity[i].amount < all_job[go_to].c_job)
            satisfied = False;
        i++;
    } /* while satisified*/
if(satisfied) done = True;
T++;
} /* while !done */

/* 15.. scheduling the job if the capacity isn't violated*/
if(satisfied){
for(o=0; o<N; o++)
    if(EDD_sequence[o].job_no == all_job[go_to].job_no)
        loc_in_EDD = o;
    all_scheduled_job[loc_in_EDD].job_no = all_job[go_to].job_no;
    all_scheduled_job[loc_in_EDD].st_job = st_counter;
    all_scheduled_job[loc_in_EDD].et_job = st_counter +
            all_job[all_scheduled_job[loc_in_EDD].job_no-1].p_job;
    all_job[all_scheduled_job[loc_in_EDD].job_no-1].status = 3;
    o_cr_job[j].status = 3;

    /* 16.. Update Capacity */
    for(L=st_counter; L < (st_counter+all_job[go_to].p_job); L++)
    {   Current_Resource_Capacity[L].amount =
                Current_Resource_Capacity[L].amount -all_job[go_to].c_job;
            } /* for L */
        } /* if(satisfied) */
    } /* for j */

    /* 17 .. find the last et of the job */
    milestone = -1;
    for(j=0; j<N; j++)
    {  if(all_scheduled_job[j].et_job > milestone)
            milestone = all_scheduled_job[j].et_job;}

    /* 18 .. Phase 4:
            sequence the unscheduled in any order after the last sequenced job from the
            previous phases*/

    for(j=0; j<o_cr_no; j++)
    {  if(o_cr_job[j].status == 1)
       {    go_to = o_cr_job[j].job_no - 1;
            satisfied = True;
            not_at_this_tp = False;

            /* 19.. scheduling disregarded jobs starting from the last ET in the
                     sequence (milestone)*/
            for(T=milestone; T<(t+1);T++)
            {    satisfied = True;
                 while((satisfied) && (!not_at_this_tp))
                 /* 20.. Continue as long as the resource can still hold the job */
                 {    for(k=T; k<(T+all_job[go_to].p_job); k++)
                      /* 21.. checking if the resource can hold the
                                  resource for the whole job processing time */
                      { if(Current_Resource_Capacity[T].amount < all_job[go_to].c_job)
                                  {satisfied = False;}
            } /* for k*/
            if(satisfied) not_at_this_tp = True;
            if(satisfied){
                 for(o=0; o<N; o++)
                      if(EDD_sequence[o].job_no == all_job[go_to].job_no)
                                 loc_in_EDD = o;
                 all_scheduled_job[loc_in_EDD].job_no = all_job[go_to].job_no;
                 all_scheduled_job[loc_in_EDD].st_job = k - all_job[go_to].p_job;
```

```
          all_scheduled_job[loc_in_EDD].et_job = k;
          all_job[all_scheduled_job[loc_in_EDD].job_no-1].status = 4;
          o_cr_job[j].status = 4;

          /* 22.. Update Capacity */
          for(L=T; L < (T+all_job[go_to].p_job); L++)
          {    Current_Resource_Capacity[L].amount =
               Current_Resource_Capacity[L].amount -all_job[go_to].c_job;
          } /* for L */
      }/* if satisfied */
  }/* while */
  } /* for T */
  } /* if o_cr_job[] */
} /* for j */

printf(“\n\n ........ Capacitated machine sequencing........\n”);
printf(“\n Current_Resource_Capacity[i] -> Disregarded_jobs - tm_period\n”);
printf(“ ------------------------------\n”);
for(i = 0;i<t; i++)
printf(“period = %d Current_Resource_Capacity = %d\n”,
          i,Current_Resource_Capacity[i]);
printf(“ -------------------\n”);
tardy_counter = 0;
unsequenced_counter = 0;
for(i = 0;i<N; i++)
{ printf(“job = %d st_job = %d et_job = %d status= %d Tardiness = %d\n”,
          all_scheduled_job[i].job_no,
          all_scheduled_job[i].st_job,
          all_scheduled_job[i].et_job,
          all_job[(all_scheduled_job[i].job_no)-1].status,
          tardy = all_job[all_scheduled_job[i].job_no-1].d_job -
          all_scheduled_job[i].et_job);
  if(tardy < 0) tardy_counter++;
  if(all_scheduled_job[i].job_no == 0) unsequenced_counter++;
  } /* for i */
printf(“\n\n ........ Conclusion ........\n”);
printf(“\n\n.../ %d job(s) attempted \n”,N);
printf(“.../ %d job(s) sequenced .... with %d tardy job(s) \n”,
      N-unsequenced_counter-tardy_counter,tardy_counter);
if(unsequenced_counter > 0)
```

```
{
  printf("../ %d job(s) not sequenced .... because the\n",unsequenced_counter);
  printf(" resource window could accommodate all jobs\n");
}
end /* CR_Disregarded_jobs_2 */
/* ----------------CR_Disregarded_jobs_2 - End------------------- */
```

C.2 Sample data file

The first line contains the number of jobs (N) and the resource time window (T). The rest of the lines contain the job number, ready time, due date, processing time and capacity requirement of each job

```
5 40
1 3 5 120
2 0 4 60
3 2 6 2 20
4 0 1 1 20
5 1 3 1 60
```

APPENDIX D Experimental results of the sequencing heuristic

A number of randomly generated test cases (110 cases) were used to test the sequencing heuristic. This appendix presents the results of these test cases.

D.1 About the test cases

The four versions of the heuristic were tested over two sets of data; in the first set (Set A)[1], the starting times were based on Poisson distribution. Two values of lambda [λ] (inter-arrival times between jobs) were used: 3.75 and 7.5 minutes. These values are based on actual data from McDonald's corporate store located in the Scarborough[2]. Lambda = 3.75 minutes is for the dinner period (5:30 pm to 8:00 pm) while lambda = 7.5 minutes is for the lunch period (11:30 am to 2:30 pm). The processing times and the capacity requirements of each job are assumed to be independent of the starting times and were drawn from a uniform distribution.

The second set (**Set B**)[3] of data consists of 109 test problems which were specially designed to explore the performance of the heuristic by varying the problem size and the capacity requirements while keeping the starting processing time constant. Seven of these test problems have a known optimal solution obtained through using CPLEX mixed-integer programming optimizer. The rest of the test problems are used to compare the four versions of the heuristic.

[1] tables D.2 through D.4.

[2] Reference to Yan, Gary, Corporate store manager, McDonald's

[3] tables D.6 through D.11.

Four versions of the heuristic were tested for the purpose of experimenting on the different choice criteria in the heuristic. In the first phase of the heuristic, the assignment of jobs to time slots is based on either:

- earliest start time or
- latest start time of each job

Moreover, in the case when having capacity violations, the job's disregarding criterion could be either:

- highest capacity requirement among the activities at the time of violation or
- highest capacity requirement and the longest processing time among the activities at the time of violation.

The four versions of the heuristic are presented in table D.1.

TABLE D.1 Four versions of the sequencing heuristic

Version	Assignment of jobs	Criteria of disregarding jobs
ESP	Earliest start time	Longest processing time
ESPR	Earliest start time	Longest processing time & Highest capacity requirement
LSP	Latest start time	Longest processing time
LSPR	Latest start time	Longest processing time & Highest capacity requirement

The experimental results are presented in tables D.2 through D.11.The tables include the number of tardy jobs generated[1]. In table D.6 the optimal solution of a number of problems is presented. The optimal results are obtained as a result of applying the integer programming formulation in CPLEX which is mixed integer optimizer developed by CPLEX Optimization Inc. However, obtaining the optimal solution using the optimizer is impractical as it takes around ten hours to solve a 40 job problem. Tables D.6 through D.8 include the solution time of a number of the

[1] For <u>data set B</u>, the tables include the duration elapsed in solving a problem using one of the four versions of the heuristic.

tests performed, while the rest of the tables specify only the number of tardy jobs as a result of using the different versions of the heuristic.

TABLE D.2 Experimental results **SET A**

$\lambda = 3.75$

#	# of jobs	Tardy jobs in ESPR	Tardy jobs in ESP	Tardy jobs in LSPR	Tardy jobs in LSP
1	31	12	15	12	15
2	54	29	34	29	34
3	48	24	30	24	30
4	44	22	27	22	27
5	40	17	20	17	20
6	46	21	27	21	27
7	50	27	27	27	34
8	41	20	34	20	25
9	58	31	25	31	40
10	40	18	21	18	21
11	44	22	30	22	30
12	31	12	14	12	14
13	64	36	41	36	41
14	50	27	33	27	33
15	52	26	33	26	33
16	51	27	33	27	33
17	38	18	23	18	23
18	44	19	21	19	21
19	46	22	27	22	27
20	60	33	39	33	39

TABLE D.3 Experimental results **SET A** - cont'd

$\lambda = 7.5$

#	**# of jobs**	**Tardy jobs in ESPR**	**Tardy jobs in ESP**	**Tardy jobs in LSPR**	**Tardy jobs in LSP**
21	18	4	4	4	4
22	24	7	8	7	8
23	32	11	13	11	13
24	29	9	8	9	8
25	27	10	11	10	11
26	24	8	8	8	8
27	27	11	12	11	12
28	19	5	6	5	6
29	24	7	7	7	7
30	32	11	13	11	13

TABLE D.4 Experimental results **SET A** - cont'd

$\lambda = 3.75$

#	# of jobs	Tardy jobs in ESPR	Tardy jobs in ESP	Tardy jobs in LSPR	Tardy jobs in LSP
31	55	26	29	26	29
32	69	37	43	37	43
33	58	29	37	29	37
34	61	29	32	29	32
35	71	39	40	39	40
36	68	36	47	36	47
37	64	32	36	32	36
38	62	29	36	29	36
39	58	29	41	29	41
40	75	42	48	42	48
41	124	82	89	82	89
42	119	73	84	73	84
43	137	89	107	89	107
44	125	78	92	78	92
45	132	82	92	82	92
46	135	86	98	86	98
47	118	73	88	73	88
48	115	68	81	68	81
49	120	76	91	76	91
50	139	90	112	90	112

TABLE D.5 Experimental results **SET A** - cont'd

$\lambda = 7.5$

#	# of jobs	Tardy jobs in ESPR	Tardy jobs in ESP	Tardy jobs in LSPR	Tardy jobs in LSP
51	48	15	19	16	17
52	64	21	25	16	24
53	51	16	19	15	18
54	69	25	29	24	27
55	58	19	21	18	20
56	68	25	36	26	33
57	50	15	15	15	16
58	56	17	22	17	21
59	74	26	32	25	29
60	69	28	34	27	29
61	63	21	24	21	22
62	56	17	19	18	23
63	60	21	23	21	22
64	60	17	24	17	19
65	66	24	28	23	29

TABLE D.6 Experimental results - **SET B**

#	# of jobs	Tardy jobs in **ESPR**	Solution time (seconds)	Tardy jobs in **ESP**	Solution time (seconds)	Tardy jobs in **LSPR**	Solution time (seconds)	Tardy jobs in **LSP**	Solution time (seconds)	optimal solution
1	10	3	0.0	2	0.0	2	0.0	1	0.0	1
2	31	1	0.0	1	0.0	0	0.0	0	0.0	0
3	31	0	0.0	1	0.0	0	0.0	0	0.0	0
4	31	0	0.0	1	0.0	0	0.0	0	0.0	0
5	32	0	0.0	1	0.0	0	0.0	0	0.0	0
6	33	1	0.0	2	0.0	1	0.0	1	0.0	1
7	34	2	0.0	2	0.0	2	0.0	2	0.0	2
8	68	6	0.2	5	0.3	5	0.2	5	0.3	☞[a]
9	136	14	0.8	11	1.1	11	0.7	11	1.0	

[a] Could not be computed using integer programming formulation. When a problem of size 40 was tested, it took an hour and thirty minutes CPU time. Actually it took more then ten hours real time. Accordingly, test problems with size greater than 40 were not tested using integer programming.

TABLE D.7 Experimental results **SET B** - cont'd

#	# of jobs	Tardy jobs in **ESPR**	Solution time (seconds)	Tardy jobs in **ESP**	Solution time (seconds)	Tardy jobs in **LSPR**	Solution time (seconds)	Tardy jobs in **LSP**	Solution time (seconds)
10	136	23	0.7	21	1.0	23	0.6	26	0.9
11	136	25	0.6	22	0.6	21	0.6	22	0.6
12	136	21	0.5	22	0.6	19	0.6	17	0.7
13	136	13	0.7	13	0.6	15	0.5	13	0.5
14	136	19	0.7	16	0.6	16	0.5	13	0.7
15	136	17	0.8	14	0.8	8	0.7	8	0.6
16	136	14	0.8	17	0.9	14	0.7	11	0.8
17	136	11	0.8	11	1.0	11	0.5	11	0.6
18	136	8	0.6	14	0.8	8	0.5	11	0.5
19	136	11	0.5	11	0.9	11	0.6	11	0.5
20	136	14	0.7	11	0.9	8	0.6	8	0.6
21	136	11	0.6	11	0.7	11	0.7	11	1.0
22	136	8	0.6	8	1.1	8	0.6	8	0.9
23	136	8	0.7	8	0.8	8	0.6	8	0.7
24	136	25	0.8	22	1.0	21	0.4	26	0.8
25	136	23	0.8	21	0.9	23	0.5	26	0.7
26	136	8	0.8	14	0.9	8	0.7	11	0.7

TABLE D.8 Experimental results **SET B** - cont'd

#	# of jobs	Tardy jobs in **ESPR**	Solution time (seconds)	Tardy jobs in **ESP**	Solution time (seconds)	Tardy jobs in **LSPR**	Solution time (seconds)	Tardy jobs in **LSP**	Solution time (seconds)
27	93	15	6.8	24	15.4	14	7.7	20	27.1
28	93	16	6.7	19	26	13	5.9	10	11.4
29	93	13	6.4	12	9.8	12	6	12	10.9
30	93	12	6.3	11	10.2	12	7	8	10
31	93	15	6.7	15	16	11	10	11	12
32	93	22	13	15	14	12	10.9	16	12.2
33	93	13	5.8	12	10	11	11	11	14.2
34	93	12	12.1	10	10.4	8	4.2	16	12.4
35	93	13	5.7	7	7.1	6	3.8	6	8
36	93	13	5.9	5	4.5	8	4	2	7
37	93	11	5.6	7	6.9	8	4.6	2	7.4

TABLE D.9 Experimental results **SET B** - cont'd

#	# of jobs	Tardy jobs in **ESPR**	Tardy jobs in **ESP**	Tardy jobs in **LSPR**	Tardy jobs in **LSP**
38	93	13	13	11	10
39	93	10	10	8	11
40	93	9	9	9	9
41	93	9	9	8	8
42	93	11	10	11	9
43	93	16	14	10	12
44	93	9	9	8	8
45	93	10	8	8	8
46	93	10	10	5	5
47	93	6	6	9	7
48	93	6	6	5	5
49	93	27	32	24	32
50	93	20	25	23	23
51	93	20	21	21	19
52	93	19	22	19	18
53	93	18	23	20	21
54	93	24	28	20	24
55	93	21	27	19	19
56	93	17	22	18	25
57	93	15	17	13	17
58	93	13	17	13	13
59	93	15	21	11	13
60	93	19	19	16	17
61	93	18	12	12	12
62	93	17	17	15	14
63	93	15	15	14	12
64	93	19	20	14	15
65	93	21	18	17	19
66	93	15	13	11	11
67	93	16	15	11	11
68	93	12	12	8	8
69	93	12	12	10	10
70	93	12	12	8	8

TABLE D.10 Experimental result **SET B** -cont'd

#	# of jobs	Tardy jobs in **ESPR**	Tardy jobs in **ESP**	Tardy jobs in **LSPR**	Tardy jobs in **LSP**
71	136	36	39	36	36
72	136	33	35	31	30
73	136	31	33	31	30
74	136	27	33	29	27
75	136	26	17	29	26
76	136	20	21	26	19
77	136	23	24	23	25
78	136	14	15	20	22
79	136	14	15	23	22
80	136	14	15	23	19
81	136	11	15	23	16
82	136	11	12	20	16
83	136	8	12	20	16
84	136	19	20	23	21
85	136	20	16	21	15
86	136	16	12	17	19
87	136	16	13	19	11
88	136	16	9	17	13
89	136	10	10	12	7
90	136	10	7	18	13
91	136	10	7	12	10
92	136	10	7	15	10
93	136	13	7	15	7
94	136	7	7	15	4
95	136	7	4	12	4
96	136	4	4	12	4

TABLE D.11 Experimental results **SET B** -cont'd

#	# of jobs	Tardy jobs in **ESPR**	Tardy jobs in **ESP**	Tardy jobs in **LSPR**	Tardy jobs in **LSP**
97	136	29	31	22	25
98	136	27	30	25	24
99	136	23	25	19	20
100	136	18	23	19	20
101	136	21	21	17	19
102	136	13	20	10	11
103	136	19	14	16	14
104	136	7	11	13	14
105	136	10	8	7	11
106	136	7	17	10	11
107	136	10	8	7	14
108	136	7	8	7	11
109	136	4	8	7	11
110	136	4	8	4	8

APPENDIX E *Activity-state, temporal axioms and data*

Throughout the resource microtheory, number of ground terms and axioms are used that are defined in the "activity-state" and time ontologies. This appendix presents these terms.

E.1 Activity-state, Temporal axioms and data

This appendix axioms that are used through put the thesis. The detailed semantical description of these relations as in TOVE manual [TOVE 92].

period_contains(Ti, Tp):- strictly_contains(Ti, Tp) | possibly_contains(Ti, Tp).

period_overlaps(Ti1, Ti2):- striclty_overlaps(Ti1,Ti2) | possibly_overlaps(Ti1, Ti2) | striclty_overlaped_by(Ti1,Ti2) | possibly_overlaped_by(Ti1, Ti2).

period_after(Ti, Tp):- strictly_after(Ti, Tp) | possibly_after(Ti, Tp).

period_before(Ti, Tp):- strictly_before(Ti, Tp) | possibly_before(Ti, Tp).

This appendix contains the time data base for the examples in chapter three. It is defined in conformity with time representation in TOVE.

Time points: *time_point(TP_ID, Min, Max).*
time_point(tp1, 12, 12).
time_point(tp2, 14, 14).
time_point(tp3, 90, 90).
time_point(tp4, 100, 100).
time_point(tp5, 150, 150).
time_point(tp6, 190, 190).

```
time_point(tp7, 8, 8).
time_point(tp8, 9, 9).
time_point(tp9, 10, 10).
```

Time Interval (Period): *time_period(Ti_ID, ST, ET, MinDur, Dur, Max Dur).*

```
time_period(pd1, 12, 14, 2, 2, 2).
time_period(pd2, 14, 16, 2, 2, 2).
time_period(pd3, 11, 13, 2, 2, 2).
```

is_related:

```
is_leaf(State,[H|T]):-
 ( ( conjuncts(H,HList); disjuncts(H,HList) ),
 is_leaf(State,HList) ); ( ( conjuncts(T,TList); disjuncts(T,TList) ),
 is_leaf(State,TList) ); ( is_leaf(State,H));( is_leaf(State,T)).

is_related(Act,State):- enables(State,Act).

is_related(Act,State):-
 ( enables(EState,Act), conjuncts(EState,SList),
 is_leaf(State,SList)); ( caused_by(Act,CState), conjuncts(CState,CList),
 is_leaf(CState,CList) ).

is_related(Act,State):-
enables(EState,Act), disjuncts(EState,SList), is_leaf(State,SList).

is_related(Act,State):-
caused_by(Act,CState), disjuncts(CState,CList), is_leaf(State,CList).
```

APPENDIX F *FOL formulations of resource ontology*

This appendix contains a list of all FOL formulation of the terms, axiom, implication (etc.) included in the resource ontology.

Role:

$$(\forall\ r, a, \mathit{role}_1, , \mathit{role}_2)\ \mathit{role}(r, a, \mathit{role}_1) \wedge \mathit{role}_1 \neq \mathit{role}_2 \supset \neg\ \mathit{role}(r, a, \mathit{role}_2) \qquad \text{(FOL 1)}$$

Division of:

$$(\forall\ r_2, r, a)\ \mathit{physical\_division\_of}(r_2, r) \supset \neg\ \mathit{functional\_division\_of}(r_2, r) \qquad \text{(FOL 2)}$$

Divisibility:

$$\forall (r, a)\ \mathit{physical\_divisible}(r, a) \equiv \forall (r_1, ro_1)\ \mathit{rknown}(r) \wedge \mathit{physical\_division\_of}(r_1, r) \wedge \mathit{role}(r_1, a, ro_1) \supset \mathit{role}(r, a, ro_1) \qquad \text{(FOL 3)}$$

$$\forall (r, a)\ \mathit{functional\_divisible}(r, a) \equiv \forall (r_1, ro_1)\ \mathit{rknown}(r) \wedge \mathit{functional\_division\_of}(r_1, r) \wedge \mathit{role}(r_1, a, ro_1) \supset \mathit{role}(r, a, ro_1) \qquad \text{(FOL 4)}$$

$\forall (r, a)\ \mathit{temporal\_divisible}(r, a) \equiv \exists\ (ti, ti_1, ti_2, tp_1, tp_2, a_1, a_2, s_1, s_2)\ \mathit{rknown}(r) \wedge$

$(\mathit{uses}(s_1, r) \vee \mathit{consumes}(s_1, r)) \wedge (\mathit{uses}(s_2, r) \vee \mathit{consumes}(s_2, r)) \wedge$

$\mathit{is\_related}(a_1, s_1)$[1] $\wedge\ \mathit{is\_related}(a_2, s_2) \wedge$

$\mathit{time\_bound}(s_1, ti_1) \wedge \mathit{time\_bound}(s_2, ti_2) \wedge$

$\mathit{activity}(a_1, \mathit{executing}, tp_1) \wedge \mathit{period\_contains}(ti_1, tp_1) \wedge$

$((\mathit{activity}(a_1, \mathit{suspended}, \mathit{tp\_end}) \wedge \mathit{tp\_end} = EP(ti_1)) \vee$

[1] is a term defined in the activity-state ontology that finds an activity is linked (related) to a state.

$$((\mathit{activity}(a_1, \mathit{completed}, \mathit{tp\_end}) \wedge \mathit{tp\_end} = \mathit{EP}(\mathit{ti}_1)) \wedge$$

$$\mathit{activity}(a_2, \mathit{executing}, \mathit{tp}_2) \wedge \mathit{period\_contains}(\mathit{ti}_2, \mathit{tp}_2) \wedge$$

$$\mathit{contains}(\mathit{ti}, \mathit{ti}_1) \wedge \mathit{contains}(\mathit{ti}, \mathit{ti}_2) \wedge \mathit{role}(r, a_1, \mathit{role}) \wedge \mathit{role}(r, a_2, \mathit{role}) \qquad \text{(FOL 5)}$$

Continuos/Discrete:

$$(\forall\, r, a)\ \mathit{continuous}(r, a) \equiv \mathit{physical\_divisible}(r, a) \qquad \text{(FOL 6)}$$

$$(\forall\, r, a)\ \mathit{discrete}(r, a) \equiv \neg\ \mathit{continuous}(r, a) \qquad \text{(FOL 7)}$$

$$(\mathit{consumption\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, u) \vee \mathit{use\_spec}(r, a, \mathit{ti}, q, \mathit{rate}\ u)) \wedge \mathit{discrete}(r, a) \supset \mathit{integer}(q) \qquad \text{(FOL 8)}$$

Unit of measurement/Measured by:

$$(\forall\, r, \mathit{unit\_id}, a)\ \mathit{measured\_by}(r, \mathit{unit\_id}, a) \supset$$

$$(\exists\, u)\ \mathit{unit\_of\_measurement}(r, \mathit{unit\_id}, u, a) \qquad \text{(FOL 9)}$$

Component of:

$$(\forall\, r_1, r_2)\ \mathit{physical\_component\_of}(r_2, r_1, a) \equiv \forall(r, r_2, \mathit{ro}_1)\ \mathit{physical\_division\_of}(r_2, r_1) \wedge$$

$$\mathit{role}(r_2, a, \mathit{ro}_1) \wedge \neg \mathit{role}(r_1, a, \mathit{ro}_1). \qquad \text{(FOL 10)}$$

$$\forall\, r_1, r_2)\ \mathit{functional\_component\_of}(r_2, r_1, a) \equiv \forall(\, r, r_2, \mathit{ro}_1)\ \mathit{functional\_division\_of}(r_2, r_1) \wedge$$

$$\mathit{role}(r_2, a, \mathit{ro}_1) \wedge \neg \mathit{role}(r_1, a, \mathit{ro}_1). \qquad \text{(FOL 11)}$$

Resource Point:

$$(\forall\, r)(\exists\, Q, \mathit{tp}, \mathit{unit})\ \mathit{rp}(r, Q, \mathit{tp}, \mathit{unit}) \equiv (\exists\, q1, q2, q3 \ldots\ldots\ldots qn, sl1, sl2, \ldots\ldots\ldots\ sln)$$
$$\mathit{rknown}(r) \wedge \mathit{rpl}(r, q1, \mathit{tp}, s11, \mathit{unit}) \wedge \mathit{rpl}(r, q2, \mathit{tp}, s12, \mathit{unit}) \wedge \mathit{rpl}(r, q3, \mathit{tp}, s13, \mathit{unit})$$
$$\wedge \ldots\ldots\ldots \mathit{rpl}(r, qn, \mathit{tp}, s1n, \mathit{unit}) \wedge Q = q1 + q2 + q3 \ldots\ldots\ldots\ qn \qquad \text{(FOL 12)}$$

Encapsulation of resource points:

$$(\forall\, r)\ (\exists\, q, \mathit{tp}, \mathit{unit1})\ \mathit{rp}(r, q, \mathit{tp}, \mathit{unit1}) \equiv (\exists \mathit{unit2}, q2)\ \mathit{rp}(r, q2, \mathit{tp}, \mathit{unit2}) \wedge \mathit{transformation}(q2, q, \mathit{unit1}, \mathit{unit2}, r)) \qquad \text{(FOL 13)}$$

Resource existence:

$$(\forall\, r)\ (\exists\, \mathit{tp})\ \mathit{rexist}(r, \mathit{tp}) \equiv \mathit{rknown}(r) \wedge (\exists\, l, q, u)\ \mathit{rpl}(r, q, \mathit{tp}, l, u) \wedge (q > 0) \qquad \text{(FOL 14)}$$

$$(\forall\, r)(\exists\, \mathit{tp}, l)\ \mathit{rexistl}(r, \mathit{tp}, l) \equiv \mathit{rknown}(r) \wedge (\exists\, q, u)\ \mathit{rpl}(r, q, \mathit{tp}, l, u) \wedge (q > 0) \text{(FOL 15)}$$

Application specifications:

$$(\exists\, a, q, \mathit{ti}, \mathit{rate}, \mathit{unit})\ (\forall r)\ \mathit{consumption\_spec}(r, a\ \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \equiv (\exists\, s, s_2, \mathit{unit\_id})$$

$$\mathit{enabling}(s, a)\ \wedge$$

$$\mathit{is\_related}(a, s_2) \wedge \mathit{consumes}(s_2, r) \wedge \mathit{quantity}(s_2, q) \wedge \mathit{time\_bound}(s_2, \mathit{ti})\ \wedge$$

$$\mathit{unit\_of\_measurement}(r, \mathit{unit\_id}, u, a)\ \wedge \mathit{measured\_by}(r, \mathit{unit\_id}, a) \qquad \text{(FOL 16)}$$

$$(\forall r, a, s, \mathit{ti}, q', \mathit{rate}, \mathit{unit})\ \mathit{consumption\_spec}(r, a, \mathit{ti}, q', \mathit{rate}, \mathit{unit}) \equiv (\forall\, s, r, a, q, \mathit{ti}, \mathit{tp}, \mathit{tp}')$$

$$\mathit{rp}(r, q, \mathit{tp}, \mathit{unit\_id}) \wedge ((\mathit{is\_related}(a, s) \wedge \mathit{consumes}(s, r)) \wedge$$

$$\mathit{tp} = \mathit{SP}(\mathit{ti}) \wedge \mathit{tp}' = \mathit{EP}(\mathit{ti}) \wedge \mathit{enabling\_state}(s, \mathit{tp}, \mathit{enabled})$$

$$\supset \mathit{rp}(r, q - q', \mathit{tp}', \mathit{unit}) \tag{FOL 17}$$

$$\begin{gathered}(\exists\, a, q, \mathit{ti}, \mathit{rate}, \mathit{unit})\ (\forall r)\ \mathit{use\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \equiv (\exists\, s, s_2, u)\ \mathit{enabling}(s, a) \wedge \\ \mathit{is\_related}(a, s_2) \wedge \mathit{uses}(s_2, r) \wedge \mathit{quantity}(s_2, q) \wedge \mathit{time\_bound}(s_2, \mathit{ti}) \wedge \\ \mathit{unit\_of\_measurement}(r, \mathit{unit\_id}, u, a) \wedge \mathit{measured\_by}(r, \mathit{unit\_id}, a)\end{gathered} \tag{FOL 18}$$

$$\begin{gathered}(\forall r, a, s, \mathit{ti}, q', \mathit{rate}, \mathit{unit})\ \mathit{use\_spec}(r, a, s, \mathit{ti}, q', \mathit{rate}, \mathit{unit}) \equiv (\forall\, s, r, a, q, \mathit{ti}, \mathit{tp}, \mathit{tp}') \\ \mathit{rp}(r, q, \mathit{tp}, \mathit{unit\_id}) \wedge (\mathit{is\_related}(a, s) \wedge \mathit{uses}(s, r)) \wedge \\ \mathit{tp} = \mathit{SP}(\mathit{ti}) \wedge \mathit{tp}' = \mathit{EP}(\mathit{ti}) \wedge \mathit{enabling\_state}(s, \mathit{tp}, \mathit{enabled}) \\ \supset \mathit{rp}(r, q, \mathit{tp}', \mathit{unit})\end{gathered} \tag{FOL 19}$$

$$\begin{gathered}(\exists\, a, q, \mathit{ti}, \mathit{rate}, \mathit{unit})\ (\forall r)\ \mathit{produce\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \equiv (\exists\, s, s_2, u)\ \mathit{enabling}(s, a) \wedge \\ \mathit{produces}(s_2, r)\ \mathit{is\_related}(a, s_2) \wedge \mathit{quantity}(s, q) \wedge \mathit{time\_bound}(s, \mathit{ti}) \wedge \\ \mathit{unit\_of\_measurement}(r, \mathit{unit\_id}, u, a) \wedge \mathit{measured\_by}(r, \mathit{unit\_id}, a)\end{gathered} \tag{FOL 20}$$

$$\begin{gathered}(\forall r, a, s, \mathit{ti}, q', \mathit{rate}, \mathit{unit})\ \mathit{produce\_spec}(r, a, s, \mathit{ti}, q', \mathit{rate}, \mathit{unit}) \equiv (\forall\, s, r, a, q, \mathit{ti}, \mathit{tp}, \mathit{tp}') \\ \mathit{rp}(r, q, \mathit{tp}, \mathit{unit\_id}) \wedge (\mathit{is\_related}(a, s) \wedge \mathit{produces}(s, r)) \wedge \\ \mathit{tp} = \mathit{SP}(\mathit{ti}) \wedge \mathit{tp}' = \mathit{EP}(\mathit{ti}) \wedge \mathit{enabling\_state}(s, \mathit{tp}, \mathit{enabled}) \\ \supset \mathit{rp}(r, q + q', \mathit{tp}', \mathit{unit})\end{gathered} \tag{FOL 21}$$

$$\begin{gathered}(\exists\, a, q, \mathit{ti}, \mathit{rate}, \mathit{unit})\ (\forall r)\ \mathit{release\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \equiv (\exists\, s, s_2, u)\ \mathit{causes}(s, a) \wedge \\ \mathit{releases}(s_2, r) \wedge \mathit{is\_related}(a, s_2) \wedge \mathit{quantity}(s, q) \wedge \mathit{time\_bound}(s, \mathit{ti}) \wedge \\ \mathit{unit\_of\_measurement}(r, \mathit{unit\_id}, u, a) \wedge \mathit{measured\_by}(r, \mathit{unit\_id}, a)\end{gathered} \tag{FOL 22}$$

Usage mode:

$$\begin{gathered}(\forall\, r, a)\ \mathit{continuous\_mode}(r, a) \equiv (\exists\, q, \mathit{unit\_id}, \mathit{rate}, \mathit{ti})\ (\mathit{rknown}(r) \\ (\mathit{use\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \vee \mathit{consumption\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit}) \vee \\ \mathit{produce\_spec}(r, a, \mathit{ti}, q, \mathit{rate}, \mathit{unit})) \wedge q \neq \mathit{rate}\end{gathered} \tag{FOL 23}$$

$$\begin{gathered}(\forall\, r, a)\ \mathit{discrete\_mode}(r, a) \equiv (\exists\, q, u, \mathit{rate}, \mathit{unit})\ (\\ (\mathit{use\_spec}(r, a, \mathit{ti}, q, q, \mathit{unit}) \vee \mathit{consumption\_spec}(r, a, \mathit{ti}, q, q, \mathit{unit}) \vee \\ \mathit{produce\_spec}(r, a, \mathit{ti}, q, q, \mathit{unit}))\end{gathered} \tag{FOL 24}$$

$$(\forall\, r, a)\ \mathit{continuous\_mode}(r, a) \equiv \mathit{continuous}(r, a) \tag{FOL 25}$$

Simultaneous use restriction:

$$(\forall\ a1, a2, r)\ simultaneous\_use\_restriction(a1, a2, r) \equiv (\forall\ s, s2, r, a, a2)\ use(s, a) \wedge uses(s, r) \wedge$$
$$use(s2, a2) \wedge uses(s2, r)\ (\neg\ \exists t)$$
$$\supset enabling\_state(s, tp, enabled) \wedge enabling\_state(s_2, tp, enabled) \qquad \text{(FOL 26)}$$

Committed to:

$$(\forall r, a, s, ti, ti_2, q, q', rate, unit)\ committed\_to(r, a, s, ti, q', unit) \wedge$$
$$(consumption\_spec(r, a, ti_2, q, rate, unit) \vee use\_spec(r, a, ti_2, q, rate, unit)) \supset$$
$$(contains(ti, ti_2) \vee equal(ti, ti_2)) \qquad \text{(FOL 27)}$$

Total committed:

$$(\forall\ r, tp, u)\ (\exists\ TQ)\ total\_committed(r, TQ, tp, u) \equiv (\exists\ pd1, pd2 \ldots pdn, a_1, a_2 \ldots a_n, q_1, q_2 \ldots q_n)$$
$$rknown(r) \wedge$$
$$committed\_to(r, a_1, pd1, q1, u) \wedge period\_contains(ti, pd1) \wedge$$
$$(committed\_to(r, a_2, pd2, q_2, u) \wedge \ldots\ldots \wedge committed\_to(r, a_n, pdn, q_n, u) \wedge$$
$$TQ = q_1 + q_2 + \ldots + q_n \qquad \text{(FOL 28)}$$

$$(\forall\ s, r, a, q, q', ti, tp, tp', u)$$
$$((use(s, a) \wedge uses(s, r)) \vee (consume(s, a) \wedge consumes(s, r)))\wedge total\_committed(r, q', tp, u) \wedge$$
$$enabling\_state(s, tp, possible) \wedge (tp = SP(ti)) \wedge$$
$$(tp' = EP(ti)) \supset total\_committed(r, q - q', tp', u) \qquad \text{(FOL 29)}$$

Has current activity:

$$(\forall\ r)\ (\exists\ act\_list, tp) has\_current\_activity(r, act\_list, tp) \equiv$$
$$\exists(ti, s, q, u, a)\ (committed\_to(r, a, s, ti, q, u) \wedge (period\_contains(ti, tp)$$
$$\wedge\ enabling\_state(s, tp, enabled). \qquad \text{(FOL 30)}$$

Availability for activities:

$$(\forall r)\ (\exists a, ti)\ available\_for(r, [a], ti) \equiv$$
$$(\forall\ tp \in ti)\ (\exists\ unit\_id, u, amt\_required, tq, q, amount_1, amount_2 \ldots amount_n, rate_1, rate_2 \ldots rate_n)$$
$$(consumption\_spec(r, a_1, ti, amount_1, rate_1, unit) \vee use\_spec(r, a_1, ti, amount_1, rate_1, unit)) \wedge$$
$$(consumption\_spec(r, a_2, ti, amount_2, rate_2, unit) \vee use\_spec(r, a_2, ti, amount_2, rate_2, unit)) \wedge$$
$$\ldots\ldots \wedge$$
$$(consumption\_spec(r, a_n, ti, amount_n, rate_n, unit) \vee use\_spec(r, a, ti, amount_n, rate_n, unit)) \wedge$$
$$amt\_required = amount_1 + amount_2 + \ldots\ldots + amount_n \wedge$$
$$(period\_contains(ti, tp) \wedge (total\_committed(r, tq, tp, unit)) \wedge unit\_of\_measurement(r, unit\_id, u, a)$$
$$\wedge$$
$$measured\_by(r, unit\_id, a)\ \wedge rp(r, q, tp, unit)\ \wedge (amt\_required \geq q - tq) \wedge$$
$$((\forall a_x \in a)\ no\_restricition(r, a, a_x, ti)) \qquad \text{(FOL 31)}$$

$$\begin{gathered}\mathit{no\_restricition}(r, a, a_x, ti) \equiv (\mathit{committed\_to}(r, a_x, s, ti_2, q, u) \wedge \\ (a \neq a_x) \wedge \mathit{period\_overlaps}(ti, ti_2) \wedge \neg(\mathit{simultaneous\_use\_restriction}(a, a_x, r)))\text{(FOL 32)}\end{gathered}$$

Available capacity:

$$\begin{gathered}(\forall\ r)\ (\exists\ r, tp\ amount, u)\ \mathit{available\_capavity}(r, tp, amount, u) \equiv (\forall\ a)(\exists\ tq, q, \mathit{unit\_id})\ \mathit{rknown}(r) \wedge \\ rp(r, q, tp\ u) \wedge \\ \mathit{total\_committed}(r, tq, tp, u) \wedge \mathit{unit\_of\_measurement}(r, \mathit{unit\_id}, u, a) \wedge \\ \mathit{measured\_by}(r, \mathit{unit\_id}, a) \wedge amount \geq q - tq \qquad \text{(FOL 33)}\end{gathered}$$

Trend:

$$\begin{gathered}(\forall\ r)\ (\exists\ tp)\ \mathit{trend}(r, tp, \mathit{decreasing}) \equiv \\ (\forall a\ \exists rate)\ (\mathit{rp\_at\_last\_tps}(r, a, tp, rate) \wedge (rate < 0.00)) \qquad \text{(FOL 34)}\end{gathered}$$

$$\begin{gathered}(\forall\ r)\ (\exists\ tp)\ \mathit{trend}(r, tp, \mathit{increasing}) \equiv \\ (\forall a\ \exists rate)\ (\mathit{rp\_at\_last\_tps}(r, a, tp, rate) \wedge (rate > 0.00)) \qquad \text{(FOL 35)}\end{gathered}$$

$$\begin{gathered}(\forall\ r)\ (\exists\ tp)\ \mathit{trend}(r, tp, \mathit{steady}) \equiv \\ (\forall A)\ (\exists\ amount, unit)\ (\mathit{rp\_at\_last\_tps}(r, a, tp, rate) \wedge (rate = 0.00)) \vee \\ \neg(\mathit{committed\_to}(r, a, s, ti, amount, unit) \wedge \mathit{period\_contains}(ti, tp)) \qquad \text{(FOL 36)}\end{gathered}$$

$$\begin{gathered}\mathit{rp\_at\_last\_tps}(r, a, tp, rate) \equiv (\exists\ q_1, q_2, tp_1, tp_2) \\ rp(r, q_2, tp_2) \wedge rp(r, q_1, tp_1) \wedge (rate = (q_1 - q_2)/(tp_1 - tp_2)) \qquad \text{(FOL 37)}\end{gathered}$$

Activity history:

$$\begin{gathered}(\forall\ r)\ (\exists act\_list, tp)\ \mathit{activity\_history}(r, act\_list, tp) \equiv (\forall\ a \in act\_list)(\exists ti, q, u, a, s) \\ (\mathit{committed\_to}(r,a,s,ti,q,u) \wedge \mathit{period\_before}(ti,tp) \wedge \mathit{enabling\_state}(s,tp,\mathit{completed})\text{(FOL 38)}\end{gathered}$$

Resource Configuration:

$$\begin{gathered}(\forall\ a_1, a_2, r)\ (\exists\ q_1, q_2, ti_1, ti_2, c_1, c_2, rate_1, rate_2)\ (\mathit{use\_spec}(r, a_1, ti_1, q_1, rate_1, u) \vee \\ \mathit{consumption\_spec}(r, a_1, ti_1, q_1, rate_1, u)) \wedge (\mathit{use\_spec}(r, a_2, ti_2, q_2, rate_2, u) \vee \\ \mathit{consumption\_spec}(r, a_2, ti_2, q_2, rate_2, u)) \wedge \mathit{resource\_configuration}(r, c_1, a_1) \wedge \neg \\ \mathit{resource\_configuration}(r, c_1, a_2) \supset \mathit{simultaneous\_use\_restriction}(a_1, a_2, r) \quad \text{(FOL 39)}\end{gathered}$$

Set-up time constraint:

$$\begin{gathered}(\forall\ r, a_2, l_2, dur, u)\ \mathit{set\_up}(r, a_2, l_2, dur, u) \equiv (\exists\ a_1, ti, q, c, tp_1, l_1, ct, lt, u_2, s_1) \\ \mathit{committed\_to}(r, a_1, s_1, ti, q, u) \wedge \\ tp = EP(ti) \wedge \mathit{resource\_configuration}(r, c, a) \wedge rpl(r, q, tp_1, l_1, u) \wedge \\ (\mathit{config\_set\_up}(r, a_1, a_2, ct, u_2) \wedge \mathit{loc\_set\_up}(r, l_1, l_2, lt, u_2) \wedge dur = ct + lt) \quad \text{(FOL 40)}\end{gathered}$$

Alternative resource:

$$\begin{gathered}(\forall\ r, a)\ (\exists\ list)\ \mathit{alternative\_resource}(r, a, list) \equiv (\exists\ s, s_2, \mathit{disjunct\_state})\ \mathit{uses}(s_2, r) \wedge \mathit{is\_related}(s_2, s) \wedge \\ \mathit{subclass\_of}(s, \mathit{disjunct\_state}) \qquad \text{(FOL 41)}\end{gathered}$$

Relation of the resource ontology with that of the activity-state:

$(\forall\ \mathit{state\_id})\ (\exists\ \mathit{tp})\ \mathit{completed}(\mathit{state\_id}, \mathit{tp}) \equiv (\exists\ r, a, \mathit{act\_list})$

$((\mathit{consume}(\mathit{state\_id}, a) \wedge \mathit{consumes}(\mathit{state\_id}, r)) \vee$

$(\mathit{use}(\mathit{state\_id}, a) \wedge \mathit{uses}(\mathit{state\_id}, r))) \wedge$

$\mathit{activity\_history}(r, \mathit{act\_list}, \mathit{tp}) \wedge \mathit{member\_of}(a, \mathit{act\_list})$ (FOL 42)

$(\forall\ \mathit{state\_id})\ (\exists\ \mathit{tp})\ \mathit{possible}(\mathit{state\_id}, \mathit{tp}) \equiv (\exists\ r, a, \mathit{ti}, \mathit{unit})\ (\mathit{consume}(\mathit{state\_id}, a)\ \vee \mathit{consumes}(\mathit{state\_id}, r)) \vee (\mathit{use}(\mathit{state\_id}, a)\ \vee \mathit{uses}(\mathit{state\_id}, r)) \wedge$

$\mathit{available\_for}(r, a, \mathit{ti}) \wedge \neg \mathit{committed\_to}(r, a, \mathit{state\_id}, \mathit{ti}, \mathit{amount}, \mathit{unit}) \wedge \mathit{period\_contains}(\mathit{ti}, \mathit{tp}) \wedge$

$\neg \mathit{activity}(a, \mathit{executing}, \mathit{tp})$ (FOL 43)

$(\forall\ \mathit{state\_id})\ (\exists\ \mathit{tp})\ \mathit{not\_possible}(\mathit{state\_id}, \mathit{tp}) \equiv (\exists\ r, a, \mathit{ti})$

$((\mathit{consume}(\mathit{state\_id}, a) \wedge \mathit{consumes}(\mathit{state\_id}, r)) \vee (\mathit{use}(\mathit{state\_id}, a) \wedge \mathit{uses}(\mathit{state\_id}, r))) \wedge$

$\neg \mathit{available\_for}(r, a, \mathit{ti}) \wedge \neg \mathit{committed\_to}(r, a, \mathit{state\_id}, \mathit{ti}, \mathit{amount}, \mathit{unit}) \wedge \mathit{period\_contains}(\mathit{ti}, \mathit{tp}) \wedge$

$\neg\ \mathit{activity}(a, \mathit{executing}, \mathit{tp})$ (FOL 44)

$(\forall\ \mathit{state\_id})\ (\exists\ \mathit{tp})\ \mathit{committed}(\mathit{state\_id}, \mathit{tp}) \equiv (\exists\ r, a, \mathit{ti})$

$((\mathit{consume}(\mathit{state\_id}, a) \wedge \mathit{consumes}(\mathit{state\_id}, r)) \vee (\mathit{use}(\mathit{state\_id}, a) \wedge \mathit{uses}(\mathit{state\_id}, r))) \wedge$

$\mathit{committed\_to}(r, a, s, \mathit{ti}, \mathit{amount}, \mathit{unit}) \wedge \mathit{period\_contains}(\mathit{ti}, \mathit{tp}) \wedge$

$\mathit{has\_current\_activity}(r, \mathit{act\_list}, \mathit{tp}) \wedge \neg \mathit{member\_of}(a, \mathit{act\_list}) \wedge$

$\mathit{activity\_history}(r, \mathit{list}, \mathit{tp}) \wedge \neg \mathit{member\_of}(a, \mathit{list})$ (FOL 45)

$(\forall\ r, a)\ (\exists\ \mathit{tp})\ \mathit{release\_completed}(r, a, \mathit{tp}) \equiv (\exists\ \mathit{act\_list})\ (\mathit{activity\_history}(r, \mathit{act\_list}, \mathit{tp}) \wedge$

$\mathit{member\_of}(a, \mathit{act\_list}))$ (FOL 46)

$(\forall\ r, a)\ (\exists\ \mathit{tp})\ \mathit{release\_committed}(r, a, \mathit{tp}) \equiv (\forall\ \mathit{act\_list})\ \mathit{has\_current\_activity}(r, \mathit{act\_list}, \mathit{tp}) \wedge$

$\mathit{member\_of}(a, \mathit{act\_List})$ (FOL 47)

$(\forall\ r, a)\ (\exists\ \mathit{tp})\ \mathit{release\_not\_possible}(r, a, \mathit{tp}) \equiv (\exists \mathit{state}, \mathit{act\_list}, \mathit{ti}, q, u)$

$((\mathit{period\_before}(\mathit{ti}, \mathit{tp}) \vee \mathit{period\_contains}(\mathit{ti}, \mathit{tp})) \supset \neg\ \mathit{committed\_to}(r, a, \mathit{state}, \mathit{ti}, q, u) \vee$

$(\mathit{has\_current\_activity}(r, \mathit{tp}, \mathit{act\_list}) \wedge \neg\ \mathit{member}(a, \mathit{act\_list}))$ (FOL 48)

$(\forall\ r, a)\ (\exists\ \mathit{tp})\ \mathit{produce\_completed}(r, a, \mathit{tp}) \equiv (\exists \mathit{state}, \mathit{act\_list})\ \mathit{activity\_history}(a, \mathit{act\_list}, \mathit{tp}) \wedge$

$\mathit{member\_of}(a, \mathit{act\_list})$ (FOL 49)

$(\forall\ r, a)\ (\exists\ \mathit{tp})\ \mathit{produce}(r, a, \mathit{tp}) \equiv \exists(s_2, s_3)$

$((\mathit{use}(s_2, a) \wedge \mathit{uses}(s_2, r) \wedge \mathit{enabling\_state}(s_2, \mathit{tp}, \mathit{not\_possible})\ ) \vee$

$(\mathit{consume}(s_3, a) \wedge \mathit{consumes}(s_3, r) \wedge \mathit{enabling\_state}(s_3, \mathit{tp}, \mathit{not\_possible})))$ (FOL 50)

$(\forall\ r, a)\ (\exists\ \mathit{tp})\ \mathit{produce\_possible}(r, a, \mathit{tp}) \equiv \forall s_2\ \forall s_3$

$((\mathit{use}(s_2, a) \wedge \mathit{uses}(s_2, r) \supset \mathit{enabling\_state}(s_2, \mathit{tp}, \mathit{possible})\ ) \wedge$

$(\mathit{consume}(s_3, a) \wedge \mathit{consumes}(s_3, r) \supset \mathit{enabling\_state}(s_3, \mathit{possible})))$ (FOL 51)

$(\forall\ r, a)\ (\exists\ \mathit{tp})\ \mathit{produce\_committed}(r, a, \mathit{tp}) \equiv (\exists \mathit{state}, \mathit{ti}, q, u, \mathit{act\_list})$

$$(committed\_to(r, a, s, ti, q, u) \wedge after(ti, tp) \wedge$$

$$has\_current\_activity(r, act\_list, tp) \wedge \neg\, member\_of(a, act\_list) \tag{FOL 52}$$

$$(\forall\, r, a)\ (\exists\, tp)\ produce\_enabled(r, a, tp) \equiv (\exists\, state, act\_list)\ has\_current\_activity(r, tp, act\_list) \wedge$$

$$member\_of(a, act\_list) \tag{FOL 53}$$

REFERENCES

[Aikins 83] Aikins, Janice S., *Prototypical Knowledge for Expert Systems*, Artificial Intelligence journal, vol. 20, 1983.

[Allen 83] Allen, J.F., *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM, volume 26, number 11, 1983, p. 832-843.

[Allen et al 92] Allen, James, Boddy, Mark, Breese, Jack, Burstein, Mark, Carciofini, Jim, Desimone, Roberto, Hammond, Chris, Lowrence, John, MacGregor, Robert, Russ, Tom, Schrag, Bob, Smith, Stephen, Tate, Austin, Wellman, Mike, Wilkins, Dave, *Knowledge Representation Spesification Language (KRSL)*, DARPA/Rome Laboratory PLanning and Scheduling Initiative, 1992.

[Baker 74] Baker, Kenneth R., *Introduction to Sequencing and Scheduling, John* Wiley & Sons Inc., NY, NY, 1974.

[Beeckman 90] Beeckman, D., *CIM-OSA: Computer Integrated Manufacturing - Open System Architecture*, International Journal of Computer Integrated Manufacturing, volume 2, number 2, 1990, p. 94-105.

[Bennett et al 92]Bennett, Malcolm, Buson, Susanna, Hicks, Gabriel, Kosanke, Kurt, Vigne, Veronique, *Characterizing the Need for Enterprise Integration*, Proceedings of the First International Conference, The MIT press, 1992.

[Blair et al 92] Blair, Paul, Guha, R.V, Pratt, Paul, *Microtheories: An Ontological Engineer's Guide*, CYC-050-92, Microelectronics and Computer Technology Corporation, 1992.

[Brachman 79] Brachman, R.J., *On the Epistemological Status of Semantic Networks*, Associative Networks: Representation and Use of Knowledge by Computers, Findler, N.V. (ed), Academic Press, New York, 1979.

[Chu & Ngai 93]Chu, Wesley W., Ngai, Patrick H., *Embeding temporal constraint propagation in machine sequencing for job shop scheduling*, AI EDAM, 7(1), 1993, p. 37-52.

[Davis et al 83] Davis, B.R., Smith, S., Davies, M., and St. John, W., *Integrated Computer-aided Manufacturing (ICAM) Architecture Part III/Volume III: Composite Function Model of "Design Product" (DES0)*, Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio, AFWAL-TR-82-4063 Volume III, 1983.

[ESPRIT 91a] ESPRIT, *Open System Architecture - Open System Architecture*, ESPRIT - Project 688 AMICE, CIM-OSA AD 1.0, 1991.

[ESPRIT 91b] ESPRIT, Research Reports, *CIM-OSA: Open System Architecture for CIM*, Esprit Consortium AMICE (eds), Springer-Verlag, Luxembourg, Federal Republic of German, 1991.

[Fadel & Fox 94] Fadel, Fadi George, Fox, Mark S., *A Resource Ontology for Enterprise Modelling*, To appear in the Third Industrial Engineering Research Conference (IERC 94), Atlanta, May 1994.

[Fadel et al 94] Fadel, Fadi George, Fox, Mark S., Gruninger, Michael, *A Generic Enterprise Resource Ontology*, Submitted to the third IEEE Transactions on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '94) West Virginia. 1994.

[Fikes et al 92] Fikes, Richard E., Finin, Tim, Gruber, Thomas, Mckay, Don, Neches, Robert, Patel-Schneider, Peter F., Patil, Ramesh S., *The DARPA Knowledge Sharing Effort: Progress Report*, DARPA Initiative, 1991.

[Fisher 78] Fisher, M.L., Jaikumar, R., *An Algorithm for the space-shuttle scheduling problem*, Ops. Res., volume 26, 1978, p. 166-182.

[Forbus 84] Forbus, Kenneth D., *Qualitative Process Theory*, Artificial Intelligence, 24, 1984 p. 86-168.

[Fox 83] Fox, M.S., *The Intelligent Management System: An Overview*, Processes and Tools for Decision Support, Sol, H.G (ed), North-Holland Publishing Company, 1983.

[Fox & Tenenbaum 91] Fox, M.S., and Tenenbaum, J.M., (1991), *Proceedings of the DARPA Knowledge Sharing Workshop*, Santa Barbara Ca.

[Fox 92] Fox, M.S, *Enterprise Integration Laboratory Research outline*, University of Toronto, Toronto, 1992.

[Fox et al 93a] Fox, Mark S., Chionglo, John F., Fadel, Fadi G., *A Common-Sense Model of the Enterprise*, Proceedings of the 2nd Industrial Engineering Research Conference (IERC 94) May 1993, L.A, California, 1993.

[Fox et al 93b] Fox, Mark S., Chionglo, John F., Barbuceanu, Mihai., *The Integrated Supply Chain Management System*, Submitted to the Industrial Engineering Research Conference (IERC 94) May 1994, Atlanta.

[Fox et al 94] Fox, Mark S., Gruninger, Michael, Yin, Zhan, *Enterprise Engineering: An Information Systems perspective*, To appear in the Industrial Engineering Research Conference (IERC 94) May 1994, Atlanta.

[French 87] French, Simon, *Sequencing and Scheduling: An Introduction to the Mathematics Of the Job-Shop*, Ellis Horwood Limited, Chichester, England, 1987.

[Garey et 89] Garey, Micheal, Johnson, David, *Computers and Intractability: A guide to the theory of NP-Completeness*, W.H Fremman and Company, San Francisco, California, Victor Klee (ed), 1979.

[Graham et al 79]Graham, R. L, Lawler, E.L, Lenstra, J.K, Kan, A.H.G.R, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of Discrete Mathematics, volume 5, 1979, p. 287-326.

[Grosof 92] Gosof, B. & Morgenstern, L., Watson, T,J. *Application of Logistic K.R to Enterprise Modelling*, AAAI Workshop on Enterprise Integration, 1992.

[Gruber 90] Gruber, Thomas R., *The Role of Standard Knowledge Representation for Sharing Knowledge-Based Technology*, KSL 90-53, Computer Science Department, Stanford University, 1990.

[Gruber 91] Gruber, Thomas R. *The Role of Common Ontology*, Principles of Knowledge Representation and Reasoning: Proceeding of the Second, International Conference, 1991.

[Gruber 93] Gruber, Thomas R., *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*, KSL 93-4, Computer Science Department, Stanford University, 1993.

[Guha et al 90] Guha, R.V., Lenat, Douglas B., *Cyc: A Midterm Report*, AI Magazine, Fall 1984, p. 32-59.

[Hama et al 92a] Hama, Toshiyuki, Hori, Masahiro, Nakamura, Yuichi, *Modelling Job Assignment Problems Based on Task Ontology*, IBM Research, Tokyo Research Laboratory, 5-11 Sanban-cho, Chiyoda-ku, Tokyo 102, Japan, RT 0076, 1992.

[Hama et al 92b] Hama, Toshiyuki, Hori, Masahiro, Nakamura, Yuichi, *Identifying Reusable Problem-Solving Knowledge as Task-Specific Components*, IBM Research, Tokyo Research Laboratory, Chiyoda-ku, Tokyo 102, Japan, RT 0078, 1992.

[Hayes 90] Hayes, Patrick J., *Naive Physics I: Ontology for Liquids*, Readings in Qualitative Reasoning about Physical Systems, Weld, Daniel S. (Eds.), 1990, p. 484-502.

[Hirst 89] Hirst, Graeme, *Ontological Assumptions in knowledge Representation*, Proceedings of the International Conference on Principles of Knowledge Representation an Reasoning, 1st, Toronto, 1989.

[IFAC/IFIP 93] Williams, Theodore J., Bernus, Peter, Chen, David, Doueingts, Guy, Nemes, Laszlo, Nevins, James L., Vallespir, Bruno, Zoetekouw, Dick, with the contributions of the other members of the task forse, *A technical report on the IFAC/IFIP task force on architecture for integrating manufacturing activities and enterprises*, 1993.

[Kim & Fox 94] Kim, Henry, Fox, Mark S., *Formal Models of Quality and ISO 9000 Compliance: An Information systems Approach*, To appear in the Proceedings of ASQC Quality Contro Congress, Las Vegas, NV, 1994.

[Kise et al 78] Kise, Hiroshi, Inbarki, Toshihide, Mine, Hisashi, *A Solvable Case of the One-Machine Scheduling Problem with Ready and Due Times*, Operations Research, volume 26, number 1, 1978.

[Kuipers 84] Kuipers, Benjamin, *Common sense Reasoning about Causality: Deriving Behavior from structure*, Artificial Intelligence, 24, 1984. pp 169-203.

[Lenstra et al 77]Lenstra, J.K, Kan, A.H.G.R, Buchers, P., *Complexity in Machine Scheduling Problems*, Annals of Discrete Mathematics, volume 1, 1977, p. 343-362.

[Mizoguchi et al 92]Mizoguchi, Riichiro, Tijerino, Yuri, Ikeda, Mitsuru, *Task Ontology and its use in as Task Analysis - Interview System*, Proceedings of JKAW 92, 1992, p.185-198.

[Moore 68] Moore, J.M, *An n-job, one machine sequencing algorithm for minimising the number of late jobs*, Mgmt. Sci., 15, 1968, p. 102-109.

[Nahmias 89] Nahmias, Steven, *Production and Operation Analysis*, IRWIN, 1989.

[Parnuk 87] Parnuk, H. Van Dyke, White, John F., *A synthesis of factory reference models*, Industrial Engineering Institute, Ann Arbor, Michigan, 1987.

[Patterson et al 74] Patterson, J.H., Huber, D., *A Horizon varying, Zero one Approach to Project Scheduling*, Management Science, volume 20, number 6, 1974, p. 990-998.

[Petrie 92] Patrie, Charles, *Enterprise Integration - Introduction*, Proceedings of the First International Conference, The MIT press, 1992, p.1-14.

[Pinto & Reiter 93] Pinto, J. and Reiter, R. *Temporal reasoning in logic programming: A case for the situation calculus.* In Proceedings of the Tenth International Conference on Logic Programming, Budapest, June 1993.

[Sathi et al 85] Sathi, A., Fox, M.S., Greenberg, M., *Representation of activity knowledge for project management.* IEEE Transactions on Pattern Analysis and Machine Intelligence. PAMI-7:531-552, September, 1985.

[Scheer 89] Scheer, A.W, *Enterprise-Wide Data Modelling*, Springer-Verlag 1989.

[Slowiniski 80] Slowinski, R., *Two Approaches to Problems of Resources Allocation Among Project Activities - A Comparative Study*, J. Operational Research Society, volume 31, number 8, 1980, p.711-723.

[Smith 90] Smith, Stephen F., *The OPIS Framework for Modelling Manufacturing Systems*, Center for integrated Manufacturing Decision Systems, The Robotics Institute, Carnegie Mellon University, Pittsburgh, CMU-RI-TR-89-30, December, 1990.

[Sommervil 92] Sommervil, Ian, *Software Engineering*, Andison-Wisley, Lancaster, England, McGettrick, A.D, Leeuwen, J. van, 1992.

[Sowa et al 92] Sowa, John F., Zachman, John A., *A Logic-Based Approach to Enterprise Integration*, Proceedings of the First International Conference, The MIT press, 1992.

[Sussenguth 92] Sussenguth, Wolfran, Jochem, Roland, *An object oriented method for integrated enterprise modelling applied for development of enterprise-related CIM-strategies and general CIM-standards*, Production Technology Centre Berlin, 1992.

[Talbot 82] Talbot, F. Brian, *Resource-Constraint Project scheduling with Time-resource Trade-offs: The Non Preemptive case*, Management Science, volume 28, 1982, p.1197-1210.

[Tham & Fox 94]Tham, Donald, Fox, Mark S., *A Cost Ontology for Enterprise Modelling*, To appear.

[Tenenbaum et al 92]Tenenbaum, Jay M., Weber, Jay C., Gruber, Thomas R., *Enterprise Integration - Lessons from SHADE and PACT*, Proceedings of the First International Conference, The MIT press, 1992.

[TOVE 92] Fox, Mark S., Chionglo, John F., Fadel, Fadi G., *TOVE manual,* University of Toronto 1992.

[TOVE 94] Fox, Mark S., Chionglo, John F., Fadel, Fadi G., Gruninger, Michael, *TOVE manual (The second veriosn),* University of Toronto 1994, To appear.

[Wegalrz et al 80] Weglarz, J., Blazewicz, J., Cellary, W., Slowinski, R., *An Automatic Revised Simplex Method for Constraint Network Scheduling*, ACM Trans. Math. Software, volume 3, number 3, 1977, p. 295-300.

[Wilkins 88] Wilkins, David E., *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan, Michael B. (ed), Morgan Kuafmann Publishers, Inc., 1988.

[PERA 91] Williams, T.J., and the Members, Industry-Purdue University Consortium for CIM, *The PURDUE Enterprise Reference Architecture*, Purdue Laboratory for Applied Industrial Control, Purdue University, West Lafayette, Report Number 154, 1991.

[Zweben et al 94]Zweben, Monte, Brian, Daun, Davis, Eugene, Deale, Michael, *Scheduling and Rescheduling with Gerry"*, To appear in "Inelligent Sceduling", Zweben, Monte, Fox, Mark S., Morgan Kuafman, 1994.