TOWARDS A NEW APPROACH FOR ENTERPRISE INTEGRATION: THE SEMANTIC MODELING APPROACH

A Thesis Submitted to the

College of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of Masters of Science

in the Department of Mechanical Engineering

University of Saskatchewan

Saskatoon, Saskatchewan

Canada

By

RANGA PRASAD RADHAKRISHNAN

PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of my material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Mechanical Engineering

University of Saskatchewan

Saskatoon, Saskatchewan

Canada S7N 5A9

ABSTRACT

Manufacturing today has become a matter of the effective and efficient application of information technology and knowledge engineering. Manufacturing firms' success depends to a great extent on information technology, which emphasizes the integration of the information systems used by a manufacturing enterprise. This integration is also called enterprise application integration (here the term *application* means information systems or software systems). The methodology for enterprise application integration, in particular enterprise application integration automation, has been studied for at least a decade; however, no satisfactory solution has been found. Enterprise application integration is becoming even more difficult due to the explosive growth of various information systems as a result of ever increasing competition in the software market. This thesis aims to provide a novel solution to enterprise application integration.

The semantic data model concept that evolved in database technology is revisited and applied to enterprise application integration. This has led to two novel ideas developed in this thesis. *First*, an ontology of an enterprise with five levels (following the data abstraction: generalization/specialization) is proposed and represented using unified modeling language. *Second*, both the ontology for the enterprise functions and the ontology for the enterprise applications are modeled to allow automatic processing of information back and forth between these two

domains. The approach with these novel ideas is called the enterprise semantic model approach.

The thesis presents a detailed description of the enterprise semantic model approach, including the fundamental rationale behind the enterprise semantic model, the ontology of enterprises with levels, and a systematic way towards the construction of a particular enterprise semantic model for a company. A case study is provided to illustrate how the approach works and to show the high potential of solving the existing problems within enterprise application integration.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere thanks to my supervisor, Professor C. Zhang, for his invaluable guidance, stimulating discussion and continuous encouragement during my whole research as well as for his critical review of the manuscript.

I would like to extend special thanks to the other members of my advisory committee: Professor S. Habibi and Professor G. Watson. Their valuable support and constructive suggestions have greatly improved the present work.

I acknowledge Professor M. Hojati and Dr. Z. Ma for guiding me during the initial stages of my research. I would like to thank Mr. G.V. Shankar, CEO of Saisu Technologies Inc., for allowing me to study its software product. I also acknowledge Dr. H.M. Morelli from the English Department of the University of Saskatchewan for reviewing the draft version of this thesis and giving me valuable suggestions in technical English writing.

My research was made possible by the generous support of Saisu Technologies Inc., Saskatoon, and the Natural Sciences and Engineering Research Council (NSERC).

Dedicated to my friends & family

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

API	Application Programming Interface
ARIS	Architecture for Integrated Information System
CAD	Computer Aided Design
CIM	Computer Integrated Manufacturing
CIMOSA	Computer Integrated Manufacturing Open Systems Architecture
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationship Management
DBMS	Database Management System
EAI	Enterprise Application Integration
EBE	Enterprise Business Entity
EI	Enterprise Integration
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planning
ESM	Enterprise Semantic Model
GEO	Generic Enterprise Ontology
GIM	Graphs with Results and Activities Interrelated Integrated Methodology
GPEO	Generic Process Enterprise Ontology
GRAI	Graphs with Results and Activities Interrelated
GSEO	Generic Steel Enterprise Ontology
HRD	Human Resource Development
HTTP	Hyper Text Transfer Protocol
IEM	Integrated Enterprise Modeling
IIOP	Internet Inter-Object Request Broker Protocol
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
JDBC	Java Database Connectivity
OCL	Object Constraint Language
ODBC	Open Database Connectivity
ORB	Object Request Broker
OS	Operating System
PERA	Purdue Enterprise Reference Architecture
PLC	Programmable Logic Controller
R&D	Research and Development
SOAP	Simple Object Access Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TOVE	Toronto Virtual Enterprise
UML	Unified Modeling Language
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 Enterprise Today

Customer demand for fast delivery and high quality functions has increased considerably since the explosion of information system usage in enterprises in the 1990s. This demand has had a profound impact on manufacturing and other service organizations, all of which will be called **enterprises** throughout this thesis. Increasing resources used to be considered a solution to this demand problem, but such a solution does not work well nowadays because increasing costs are beyond what customers will accept. Another problem with this solution is related to a new concept of economic development called 'sustainable' development. In short, sustainable development advocates the need for long-term planning of resources, which can conflict with this solution of increased resources. Another solution could be to advance the technology of enterprises. It is usually the case that new technology development has risk of failure and has long lead-times. Thus, there is a need to study alternative solutions. The philosophy of any new solution should be to maximize efficiency with minimal and sustainable utilization of resources. A straightforward solution is 'lean production,' [Warnecke & Huser 1995] which was developed in the 1980s. However, the lean production concept has been shown not to improve productivity in terms of fast delivery of products/services and in terms of its ease in adapting to change [Green 1999]. Considerable efforts were taken to

find further new solutions; among others, computer integrated manufacturing **(CIM)** and concurrent engineering approaches appear to be most effective.

The CIM and concurrent engineering concepts advocate a large-scale use of information technology in enterprise. As a result, computer program systems, including databases and programs, have been developed with the objective of providing support to every functional unit of an enterprise, for example, designing, fabricating, shop-floor planning, and finance. The CIM concept has subsequently created or stimulated some business of developing technical and business software because enterprises could not afford to develop and maintain their own proprietary systems. Examples of some giant software companies are Oracle™ (for database management) [Oracle 2003], Baan™ (for enterprise resource planning) [Baan 2003], and i2™ (for supply chain management) [i2 2003].

Parallel to the development of CIM and concurrent engineering, another concept called **virtual enterprise** [Presley et al. 2001] was proposed in the early 1990s. Primarily, the virtual enterprise principle says that enterprises should concentrate on their core competency and outsource all other functions [Nayak et al. 2001]. By forming a virtual chain of enterprises, the changes made in one enterprise do not negatively affect other enterprises in the chain and, hence, the enterprises are more agile to meet changing market demands.

It is reasonable to characterize the current enterprise as (1) being information or data intensive, (2) adapting virtual enterprise as a principle for organizing and managing resources, and (3) requiring a massive amount of communication with its suppliers and customers.

1.2 Motivation

There are problems that hinder effective implementation of the modern enterprise characterized in the preceding discussion. These problems can exist at the functional level, the physical level, and the information system level. At the functional level, supplier partners or customer partners may not be identified correctly for an enterprise's particular functional needs or their actual performances may differ significantly from those required. At the physical level, products or parts supplied from suppliers may not match an enterprise's particular product environment. At the information system level, partners could have their own information systems that are different from those of the master or some other enterprise. Furthermore, these differences could be syntactic or semantic. Rapid evolution of information systems or computer programs from third parties (i.e., those technical and business software providers or vendors) may further complicate problems encountered at the information system level. Quite often, there is uncertainty about whether an enterprise's current information systems would be able to communicate with the information systems of its partners, after either a change of partners or an updating of information systems that are currently operated by the enterprise. These problems are put together and named the 'enterprise application integration (**EAI**) problem' [Lutz 2000].

Many solutions have been proposed to tackle the above problems. These solutions can be put into two categories. The first category of solutions focuses on developing methods or tools to facilitate communication of information or data between enterprises. The second category of solutions is based on the rationale that a model of an enterprise is a key towards an effective solution and focuses on modeling an enterprise's functions and resources [Lim et al. 1997]. The solutions in this category do not explicitly represent or model the enterprise's information systems. In general, the solutions from both the categories have not solved the enterprise integration problem to its entirety. A brief analysis to confirm this observation will be given in the next section, and a further detailed analysis is given in Chapter 2.

This thesis aims to develop a new solution or a new solution concept for the **EAI** problem [Cadarette & Durward 2001]. The term integration here is meant for the establishment of effective, efficient, and automated communications amongst enterprise applications within different partner enterprises. The intended research has great significance, which is evidenced by the fact that one of the Gartner-Group's top-10 predictions for 2002 [McCoy et al. 2002] is that during 2002, leading-edge businesses will exploit application integration to generate business innovation. Further evidence is the finding made by a new Hurwitz study [Migliore 2001] that of about 600 enterprises, only 10 percent have fully integrated even their most strategic business processes.

1.3 Solutions to the EAI Problem: a Brief Analysis

As discussed in the preceding section, there are two categories of solution to the EAI problem. In the following, a brief analysis is given for the purpose of deriving the research objectives of this thesis, while a more detailed analysis will be discussed in Chapter 2.

1.3.1 Solutions in the first category

Solutions in the first category can be considered to focus on information systems within enterprises. Solutions in this category include those of computing businesses looking for means by which two databases or two programs can communicate with each other. Well known proposals include (1) **CORBA**™ (Common Object Request Broker Architecture) [CORBA 2002], (2) **EJB**™ (Enterprise Java Beans) [EJB 2002], and (3) Microsoft **BizTalk**™ server [BizTalk 2002].

CORBA™ uses the standard protocol IIOP (Internet Inter-ORB Protocol), which is operating system-independent to make two enterprise applications interoperate with each other. The basic idea underlying CORBA™ is that each enterprise application

is programmed based on the CORBA™ protocol, a so-called CORBA™-based program. Because CORBA™ makes applications independent of operating systems, CORBA™ allows enterprise applications to run on different operating systems yet communicate with each other. CORBA™, however, requires all information systems to work under the same server. **EJB**™ goes beyond CORBA™ in the sense that EJB™ aims to enable two information systems supported by different servers to interoperate with each other. EJB™ takes the idea from the Java™ program, which is known to run over the Internet across different computing platforms at different levels of program system architecture. **BizTalk**™ is a middleware product for application integration. It is a collection of tools and services that performs the task of changing enterprise applications into a standard format through which the applications are able to communicate with each other. Tools that fulfill this task are called **adapters**.

ARIS (ARchitecture of Integrated Information System) [Williams 2000] takes a different approach and concentrates on the issue of the design of enterprise information systems. ARIS models all the software engineering activities including requirements, design, and implementation. ARIS has various views of information systems and various levels of abstractions. ARIS is supposed to work as follows: a particular enterprise application falls into a particular view and a particular abstraction level in the ARIS model.

In general, the solutions above do not address the semantics of enterprise functions that various applications are supposed to fulfill and do not provide a path that one can walk through from enterprise function units to enterprise information systems. It is the observation of the author of this thesis that changes affecting information systems of an enterprise could come from information systems themselves (e.g. software updating by a software company) and also from enterprise function units (e.g. enterprise reorganization, change of partners). In short, the solutions in this category do not provide an end solution to the EAI problem.

1.3.2 Solutions in the second category

Solutions in the second category take a systems engineering approach by viewing an enterprise as a system and explicitly modeling its enterprise functions and activities. Such a model helps users perform a systematic procedure to 'design' an enterprise to meet changes [Whitman & Huff 2001]. The following are several well-known studies falling into this category.

The **CIMOSA** (Computer Integrated Manufacturing Open Systems Architecture) modeling methodology [CIMOSA 1996] models the life cycle activities of an enterprise from its requirements definition to its maintenance. CIMOSA facilitates evaluating operational alternatives and decision-making. The **PERA** (Purdue Enterprise Reference Architecture) modeling methodology [Rathwell 2001] supports and guides the development of a master plan for an enterprise business entity. The business entity may be either part of a larger entity or a complete enterprise itself. PERA covers life cycle activities from identification of the business entity to its operation and maintenance. **GRAI** (Graphs with Results and Activities Interrelated) modeling methodology models the decisional structure of an enterprise and supports the design of CIM systems leading to **GIM** (GRAI Integrated Methodology) [Doumeingts & Chen 1996]. GIM covers the enterprise life cycle from the requirements to the implementation. The **IEM** (Integrated Enterprise Modeling) methodology [Lin 1999] supports the enterprise life cycle from the requirements definition to the implementation.

In general, the solutions above do not represent enterprise applications explicitly. Furthermore, these models of an enterprise do not represent the partners, especially their roles in the function chain of an enterprise. Zhang and his colleagues observed the need of representing partners in the data structures for enterprises [Zhang et al. 1999]. The main purpose of their work was to design an enterprise. They viewed an enterprise as a dynamic system in which partners are part of the system. They have,

however, not addressed the EAI problem [Gosain & Thillairajah 2002 and TechMetrix research 2002].

1.4 Statement of a New Approach: Semantic Model Approach

From the preceding discussions, it is claimed that there is a need to develop a new solution or approach to the EAI problem. This new solution must address the shortcomings in the existing solutions (i.e., the two categories of solutions discussed previously). Furthermore, the new solution must have the following elements:

(1) a model of enterprise in the aspect of resources and functions which include partners (suppliers and customers);
(2) a model of enterprise applications; and
(3) a model of the relationship between enterprise applications and enterprise functions.

One of the basic ideas that has led to the above conceptual solution is based on the successful experience of database systems as a role in integrating program files in the early 60s, as shown in Figure 1.1. In Figure 1.1(a), each program (P1 and P2) handles its own files, F1 and F2, respectively. There is no way that communication between files and programs can be made by another general purpose program. The files F1 and F2 actually represent the real world discourse. In this approach, the maintenance of a program system for any change in either program or file is done manually. This will create complexity in keeping information consistent in F1 and F2. There is another problem with this approach; the contents in the files may be redundantly represented, which may further create sources of inconsistencies in information. The database concept integrates all files by viewing contents of files as representation of the semantics of the real world discourse that the particular programs are to deal with and provides a common syntactic means of actually

writing the contents (see Figure 1.1(b)). As such, the maintenance of information consistency among program files is now taken over by a database management system. Figure 1.1(c) shows another approach. In this approach, the conceptual model captures the semantics of a real world discourse more accurately and is generically free from the constraints brought about by the syntactic problems with different DBMSs. With this approach, database designers are more transparent to the underlying DBMSs. From the point of view of integration, one could view the semantic model as a means of facilitating communications among DBMSs/ databases.

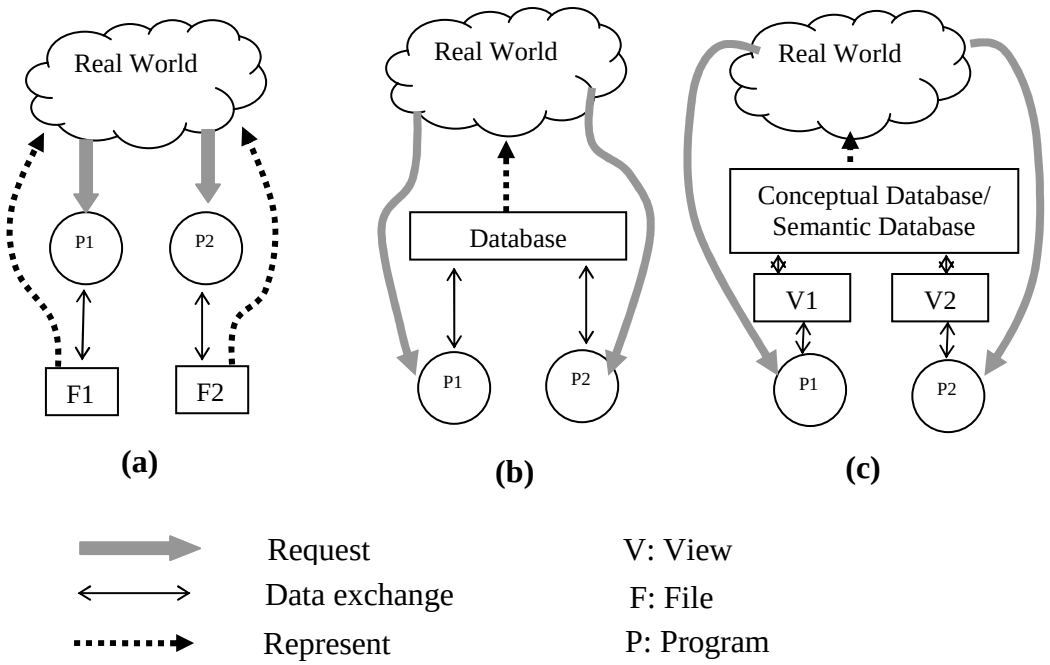


Figure 1.1 Evolution from file to semantic database

(a) The concept of program file
(b) The concept of database
(c) The concept of conceptual or semantic database

The success of the database concept as a means of integrating program files would result in a solution to the EAI problem based on the following analogy:

Real-world ⟷ Enterprise

Programs ⟷ Enterprise Applications

Database ⟷ Enterprise Semantic model (ESM)

DBMS ⟷ DBMS for ESM

This analogy can thus imply the situation shown in Figure 1.2, which is similar to the situation illustrated in Figure 1.1(c). A detailed discussion of this new solution to the EAI problem will be presented in Chapter 4.

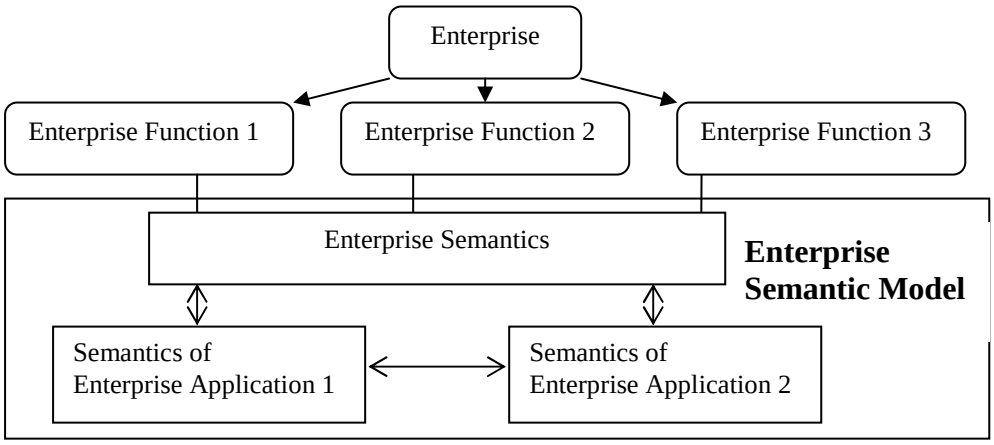


Figure 1.2 Enterprise semantic model (ESM) concept

1.5 Research Objectives, Scope and Significance

This thesis aims to develop a new solution to the EAI problem through semantic modeling. This new solution should address all the problems with the existing solutions reported in the literature. The enterprise where the new approach is applied must take the virtual enterprise as its organization principle. The following objectives are considered relevant to achieving this aim.

Objective 1: *To justify further the semantic modeling approach for the EAI problem through an analysis of the need for enterprise integration or enterprise application integration and the shortcomings of the existing approaches.*

The methodology applied to achieve this objective is to conduct an in-depth literature study and also to conduct an industrial survey. It is noticed that part of research related to this objective was described in Section 1.4.

Objective 2: *To formulate an enterprise semantic model framework. The framework needs to demonstrate its genericity and its capability of solving the problems with existing approaches.*

A framework of a model or a system is defined as a set of concepts or notions with which a concrete model or system can be built. A framework must capture the most generic things and their relationships. A framework may contain templates (e.g., databases or knowledge bases) and tools (i.e., software programs) to facilitate the building of a concrete model for a real-world application. The enterprise semantic model framework, with which this thesis is concerned, must include the ontology, generic functions and resources of enterprises, generic enterprise applications, and links between enterprise applications and enterprise functions/resources. This thesis does not, however, attempt to provide a complete model (e.g., enterprise ontology); instead, the thesis follows the research methodology shown in Figure 1.3. In Figure 1.3, the process on the left side shows the derivation of axioms/fundamental concepts. The process on the right side shows that an enterprise semantic model framework will be created. An important point here is that the semantic model is extensible.

This thesis is not going to develop any software, though in the mind of the author this is desired from a practical viewpoint. Nevertheless, the result of the thesis work will be a foundation for any kind of scenario within which a concrete product, such as software, can be developed. For instance, it is likely that a kind of software that

would help a company to diagnose its current EI or EAI problems and make recommendations on a re-engineering process for solving identified problems could be created.

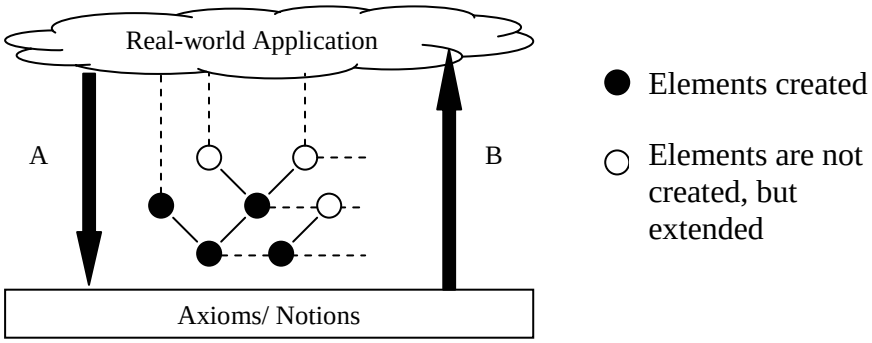


Figure 1.3 General methodology for building enterprise semantic model framework

A: a process of organization of primitives of a model: The process is data abstraction called generalization [Eriksson & Penker 2000]
B: a process of building a model: The structure shown in the middle means that the model must be created to be extensible in both horizontal and vertical directions

Objective 3: *To develop a showcase for an enterprise semantic model and to demonstrate the effectiveness of the enterprise semantic model as a potential solution to the EI or EAI problem.*

This will involve a case study. The semantic model will be instantiated for an existing enterprise. A few test cases will be developed to evaluate the proposed solution to tackle the EAI problem.

1.6 A Remark on General Research Methodology

Two types of scientific research approaches can be identified in the methodological literature, namely, the theory-developing or analytic research approach and the

design-oriented or applied research approach [van Stekelenborg 1996]. The differences in these approaches arise from the purpose of the research and the techniques that are used and not from the structure and the methods that are applied. Theory-developing research is descriptive as it describes, explains, and predicts phenomena. Design-oriented research is prescriptive, focusing on the guidelines and procedures actually needed to change phenomena. A validation of the soundness of design-oriented research is done by means of case studies. The research work described in this thesis is of the design-oriented type. This also implies that the result of the research is an artefact, which is imperative, normative, and prescriptive [van Stekelenborg 1996].

1.7 Organization of the Thesis

Chapter 2 provides a literature review to further enable the reader to understand the related subjects. It also gives further justification of the significance of the proposed work, particularly with respect to the research objectives set up in Chapter 1.

Chapter 3 introduces unified modeling language (**UML**) with special emphasis on modeling data and processes. UML will be used as a vehicle for describing a data or process model.

Chapter 4 presents a semantic model framework. This will include a more detailed discussion of the requirement for the framework, the basic elements of the framework, and a few templates.

Chapter 5 shows an instantiated semantic model for a particular enterprise, ABC in this case. It also presents a case study to show potential applications of the proposed semantic model.

Chapter 6 summarizes and concludes the thesis and addresses future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The purpose of this chapter is to provide further justification regarding the significance of the research described in Chapter 1, particularly with respect to the research objectives. A critical review of the related work reported in the literature will fulfill this purpose. This chapter also provides the background for facilitating subsequent discussions. In this chapter, the two categories of solutions or approaches to the EAI problem introduced in Chapter 1 will be further elaborated, on which the related work in literature will be reviewed. Section 2.2 will discuss enterprise modeling, which is the core of the second category of solutions to the EAI problem. This includes an in-depth discussion of the motivation and the characteristics of different enterprise modeling approaches and a comparison of them. An enterprise modeling methodology which belongs to the first category of enterprise integration solutions focusing on information systems is also discussed in this section. Section 2.3 discusses different strategies for software integration, which forms the first category of solutions to the EAI problem. This section also discusses middleware and adapter systems, which are alternative solutions to EAI. In Section 2.4 the different application integration products available in the market are analyzed and compared. Section 2.5 contains general concluding remarks.

2.2 Enterprise Modeling

2.2.1 Basic concepts

A ***model*** is the representation of a thing that could be a real world entity, a process, or an event. Because any thing has three aspects, structure, behavior, and function, there are at least three models corresponding to a thing. A process to develop a model is called ***modeling***. The purpose of modeling is to explore the structure, behavior, and function of a thing and the relationships among them.

An enterprise has structure, behavior, and function. ***Enterprise modeling*** refers to the creation of a model of an enterprise from the above mentioned three aspects [Christensen et al. 1995]. Enterprise modeling from the aspect of **structure** involves modeling the organization of resources and their relationships. The resources include people, materials, money, information systems, and equipment. Enterprise modeling from the aspect of **function** involves modeling what the enterprise intends to accomplish. Enterprise modeling from the aspect of **behavior** involves modeling how the business entities interact with each other to achieve their functions. These three aspects of the enterprise model are related as shown in Figure 2.1.

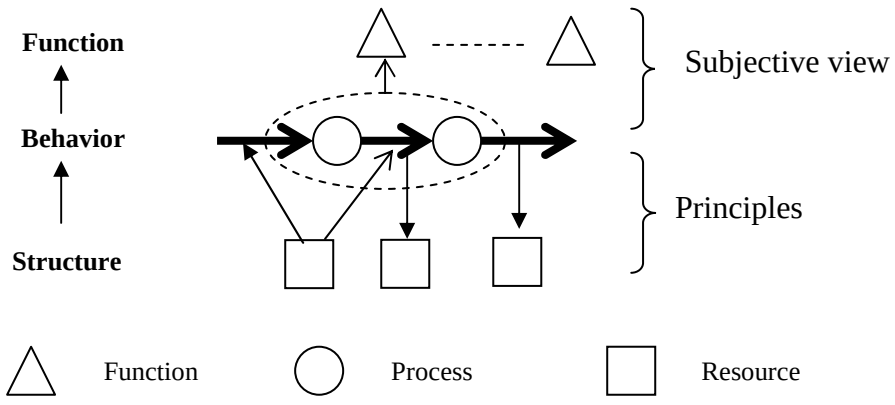


Figure 2.1 Architecture of enterprise model concept

It is shown in Figure 2.1 that structure is a basis for behaviors (processes) governed by principles. Further, function is the concept of utilization of behaviors in a meaningful manner. Examples of functions would be ‘design,’ ‘planning,’ etc. Examples of resources would be ‘steel materials,’ ‘three skilled operators,’ etc. Examples of processes would be ‘machining,’ ‘assembling,’ etc. More detailed elaborations on the concept of function, behavior and structure are found in [Eriksson & Penker 2000].

Berio and Vernadat [1999] elaborated that an enterprise model must describe:

- Three fundamental types of flows within or across enterprises:
  - Material flows (physical objects such as products, tools, raw materials);
  - Information flows (documents, data, computer files); and
  - Decision/control flows (sequence of operations).
- Five modeling views:
  - Function view: addressing enterprise functionality (what has to be done) and enterprise behavior (in which order work has to be done);
  - Information view: addressing what are the objects to be processed or to be used;
  - Resource view: addressing who or what does what;
  - Organization view: addressing organization units and their relationships, i.e., who is responsible for what or whom; and
  - Business rule view: addressing all the constraints or rules.
- Three modeling levels:
  - Requirement definition: to represent ‘the voice of the users,’ i.e., what is needed, expressed in a detailed and unambiguous way;
  - Design specification: to define formally one or more solutions satisfying the set of requirements, to analyze their properties and to select the ‘best’ one; and
  - Implementation description: to state in detail the implementation solution taking into account technical and physical constraints.

It is clear that Berio and Vernadat's argument is consistent with the function-behavior-structure architecture as shown in Figure 2.1. In later discussions related to enterprise modeling and enterprise ontology, Berio and Vernadat's architecture will be applied.

Another key concept regarding enterprise modeling is called the 'generic enterprise model' [Fox 1993]. The enterprises, irrespective of business factors, must have commonalities. The philosophy of generic enterprise model is that these commonalties should be captured and standardized into a model. The generic enterprise model can then be re-used so that modeling does not have to start from scratch every time [Szegheo 2000]. Generic enterprise modeling is consistent with the idea of modeling of systems in that it is comprised of two steps. The first step is to create a generic enterprise model. Because of its nature to capture commonalities among individual real world systems, enterprises in this case, the generic enterprise model is a template or type from the point of view of data modeling. The second step is to instantiate the template with specific semantics derived from individual systems.

2.2.2 Different schools of enterprise modeling approaches

Enterprise modeling approaches can be categorized into different schools based on their purposes. The following is a brief overview of different schools and their respective enterprise modeling approaches.

- **School 1:** *Modeling of enterprise information systems*

This school concentrates on modeling the life cycle of enterprise information systems. It helps in the software engineering of the enterprise. It provides a rich set of constructs for information modeling. It is mainly concerned with IT resources which are described from the viewpoints of control, information, and organization.

Hence this school is in the first category of enterprise integration approaches focusing on enterprise information systems.

ARIS (ARchitecture for Information Systems) [Williams 2000] focuses on the design of enterprise information systems. Therefore it provides specific modeling support for the information technology (IT) of the enterprise. ARIS supports enterprise modeling from an operational concept and IT concept to IT system implementation. It provides various views of the IT systems such as a control view, an organization view, etc.

- **School 2:** *Modeling of enterprise entities and activities*

This school concentrates on modeling the life cycle of enterprise functional entities and activities. The life cycle extends from concept creation to maintenance. Its main goal is to avoid ambiguity when a new element or person is added to the enterprise, so that the new element can be integrated well into the enterprise. Different approaches in this school are briefly discussed below.

- **CIMOSA (Computer Integrated Manufacturing Open Systems Architecture)** [CIMOSA 1996] is intended to be used for operational support rather than as a project guide in developing or re-engineering business entities. Operational uses are understood as decision support for evaluating operational alternatives as well as model driven operation control and monitoring. CIMOSA supports the engineering of an enterprise. It models the enterprise life cycle from requirements definition to implementation description. It also supports the operational use and maintenance of the models.

- **GRAI/GIM (Graphs with Results and Activities Interrelated/GRAI Integrated Methodology)** [Doumeingts & Chen 1996] was initially developed to model the decisional structure of a manufacturing enterprise for strategic, tactical and operational planning. GRAI was extended to support the design of computer

integrated manufacturing (CIM) systems leading to GRAI integrated methodology (GIM) as an integrated methodology for business process modeling. With special emphasis on the decisional aspects, the concept (analysis), structure (user oriented design), and realization (technical oriented design) phases of the life-cycle concept are supported.

- **IEM (Integrated Enterprise Modeling)** [Lin 1999] supports the creation of enterprise models for business re-engineering and therefore allows for the modeling of process dynamics for an evaluation of operational alternatives. IEM supports the main phases of the enterprise life cycle (requirements, design, implementation, and model up-date).

- **PERA (Purdue Enterprise Reference Architecture)** [Rathwell 2001] is intended to support and guide the development of the master plan for an enterprise business entity. The methodology covers the complete project of introduction, implementation and operation of an enterprise business entity, which may be either part of a larger entity or be a complete enterprise itself. The life cycle starts with a definition of the Business Entity to be modeled, identifying its mission, vision, management philosophy, mandates, project sponsors, leaders and members and ends with obsolescence of the plant at the end of the operational phase.

▪ **School 3:** *Enterprise ontology modeling*

This school takes the philosophy that integration means effective communication among different constituent elements. Effective communication can only be made possible through a common context including (1) sharable concepts and their representations and (2) sharable knowledge which is in the form of either what-is or how-to-do. A representation approach in this school is TOVE.

- **TOVE (TOronto Virtual Enterprise)** [Fox & Gruninger 1998, Fox et al. 1993] aims to create a generic, reusable enterprise data model that provides shared terminology for the enterprise that each agent can jointly understand and use. It defines the meaning of each term in a precise manner. It implements the semantics in a set of axioms that will enable TOVE to deduce automatically the answer to many "common sense" questions about the enterprise [Fox & Gruninger 1994].

2.2.3 A further comparison of different enterprise modeling approaches

In this section six different enterprise modeling methodologies are compared. The aspects used for comparison are (i) life cycle dimension, (ii) model view dimension, (iii) genericity dimension, and (iv) modeling language/construct used. The **life cycle dimension** includes identification of the Business Entity, definition of concepts, requirements definition, design, implementation, and operation and maintenance (see table 2.1). The **model view dimension** includes functional view, information view, decision view, organization view, and structure view (see Table 2.2). The **genericity dimension** includes the particular model and the reference architecture, which supports model creation (see Table 2.3). The **modeling language/construct dimension** includes sets of generic building blocks to represent enterprise processes, activities, information, resources, and organization. It is noted that a similar, yet slightly different extent comparison was presented by Kosanke [1996]. Further, some comments on this comparison are presented in the following.

Table 2.1 Modeling framework comparison: life-cycle (modeling levels)

	Identification	**Concept**	**Requirement**	**Design**	**Implementation**	**Operation and Maintenance**
ARIS	not defined	not defined	Operation Concept	IT System Concept	Implementation	not defined
CIMOSA	not defined	not defined	Requirement Definition	Design Specification	Implementation Description	Operation, Model maintenance
GRAI /GIM	not defined	not defined	Concept Level Analysis	Structure Level: User Oriented Design	Realization Level: Technical Oriented Design	not defined
IEM	not defined	not defined	Requirement Definition	System Design	Implementation Description	Model Update
TOVE	Meta-ontology	Ontology for Entity, relationship, role, and actor	Strategy ontology: Purpose	Strategy ontology: strategy, assumption, and Organization	Activity and process ontology: activity and resource	not defined
PERA	EBE Identification (Enterprise Business Entity)	EBE Concept Layer	EBE Definition Layer	EBE Specification layer and detailed design layer	EBE Manifestation Layer	EBE Operation Layer

In Table 2.1, PERA and TOVE cover the two uppermost layers of enterprise life cycle, i.e., the identification of the business entity and definition of concepts. This information is assumed to be provided by enterprise management in all other

methodologies. The enterprise operation is defined only in PERA and CIMOSA. Model maintenance is explicitly identified in both CIMOSA and IEM, and contained in the operation layer of PERA. ARIS and GRAI only support the requirements, design, and implementation aspects. In addition, ARIS covers the life cycle of information systems.

Table 2.2 Modeling framework comparison: model views

	Function	**Information**	**Decision/ Organization**	**Structure/ Resource**
ARIS	Function View (static), and Control View (dynamic)	Data View	Organization View	Resource View
CIMOSA	Function View (static), and Function View (dynamic)	Information View	Organization View	Resource View
GRAI/ GIM	Function View (static), and Information Technical View	Information View	Decision View, Physical View, and Organization View	Physical View, and Manufacturing Technical View
IEM	Function Model View	Information Model View	not defined	not defined
TOVE	Activity and Process ontology	not defined	Organizational ontology	Resource ontology
PERA	Information Architecture (dynamic not defined)	not defined	Human and Organization Architecture	Manufacturing Architecture: Manufacturing Equipment Architecture

In Table 2.2, CIMOSA assumes one consistent enterprise model in which particular views are provided for the user in the engineering environment to allow for model engineering on a particular aspect of the enterprise operation. ARIS provides a similar approach, but has identified the control view for integrating the different views into a common process model. GRAI/GIM and PERA identify different

views, but as yet there is no real integration into one consistent model. The information view of PERA is not well defined. GRAI/GIM identifies a unique decision view, which is at the centre of the GRAI methodology, enabling modeling of strategic, tactical, and operational planning. IEM does not define model views explicitly but provides viewpoints on a common model. In TOVE the different ontologies provide for different views.

Table 2.3 Modeling framework comparison: genericity levels

	ARIS	***CIMOSA***	***GRAI/ GIM***	***IEM***	***TOVE***	***PERA***
Generic	Generic	Generic		Generic	Generic enterprise model	not defined
Partial	Reference Models	Partial	4 Levels of Abstraction	Reference	not explicitly defined	not defined
Particular	Particular	Particular		Particular	Deductive enterprise model	not defined

In Table 2.3, except for PERA, which only provides a single task module, all the other methodologies have a rather populated generic level and provide sets of partial/reference models.

From the dimension of modeling language/construct, only CIMOSA has a vision of an executable model for operation control and monitoring. TOVE uses a formal language to model all the ontologies within the enterprise. The modeling languages of CIMOSA and TOVE are very expressive. All the other methodologies use specialized languages for particular modeling purposes, e.g., language for project description (PERA), language for decision systems and CIM systems design (GRAI/GIM), languages for information systems design (ARIS), and language for business process re-engineering (IEM). Furthermore, PERA only provides textual description for modeling a task and its information inputs and outputs.

2.2.4 Special notes on methods and tools for modeling

Methods are viewed differently from tools, that is, tools are computer program systems that implement methods and have certain usability. **ARIS** has view-specific modeling methods for computer-based modeling, based on the ARIS framework. This includes extended entity relationship modeling as well as process chain diagrams, stimulus-response chains, and functional and organizational hierarchy charts. There are various commercial tools that provide comprehensive computer support for the analysis, planning and introduction of information systems. **CIMOSA** has both formal textual descriptions and graphical forms. Some of the tools that support the CIMOSA method are *FirstSTEP*™, *CimTool*™, *and PACE*™. **GRAI/GIM** has its own graphical method that is applied in the decisional model, while the IDEF0™ is used in functional modeling. A tool is being developed for implementing the GRAI/GIM method, known as *Computer Aided GIM* ™. **IEM** methodology has its own method, the IEM modeling method. The tool used to support the IEM method is called *MO2GO*™. **PERA** has its own modeling method. PERA does not mention specifically any modeling tools, but some tools like *Metis*™ and other generic modeling tools are used to support the method. **TOVE** uses a formal modeling method, 'First Order Logic,' to model the different ontologies. Some of the tools used to support the TOVE method are the *C++* environment and the *Rock* knowledge representation tool.

2.2.5 Concluding remarks

A comprehensive comparison has shown that none of the methodologies along with their tools, can provide a complete solution to the EAI problem. There are no possible consistent and rational answers to the following questions: "Does an enterprise need to purchase a particular application software called 'A', or a different one, or none?" and "What will be the impact to the software applications and business operations existing in an enterprise if 'A' is introduced?"

The enterprise modeling approach is helpful for achieving EAI by answering the second question (partially), as enterprise functions are systematically represented. Enterprise ontology modeling is an essential method for developing a knowledge system for making intelligent decisions regarding enterprises. However, a usable language for ontology modeling is not available. At this point, a powerful language called unified modeling language (UML) [Zrnec et al. 2001] will be considered in this thesis. Detailed information about UML can be found in Chapter 3.

2.3 Software Integration Solution

The first category of solutions to the EAI problem takes the philosophy that enterprise applications are computing software by their very nature. Consequently, provision of more generic ways for software systems to be able to communicate with each other can be a solution to the EAI problem [Cummings & Hanson 2002].

2.3.1 Strategies for software integration

It is noted that enterprise applications are not implemented on one operating system using one computing language. The individualism of enterprise applications has to be recognized. Under this scenario, software integration can take the following strategies [Linthicum 2001] and these strategies are shown in Figure 2.2.

- ***Data oriented:*** This focuses on the equivalence mapping between two databases, both in syntax and semantics. This approach does not change the application code that makes use of data in databases.
- ***Application-interface oriented:*** This refers to the leveraging of the interfaces exposed by applications. Developers leverage these interfaces to access both business processes and simple information. In order to integrate the applications, the interfaces must be used to access both process and data, extract the information, place it in a format understandable by the target application, and

transmit the information. Message broker (discussed in Section 2.3.2) is the perfect solution for this type of software integration strategy.

- ***Method oriented:*** This is the sharing of the business logic that exists within an enterprise. The applications may access each other's methods without having to rewrite each method within the respective application. The solutions include distributed objects, application servers, and transaction processing monitors (discussed in Section 2.3.2).
- ***Portal oriented:*** This approach is used by application architects to integrate applications by presenting information from several local or partner applications within the same user interface.
- ***Process integration oriented:*** This provides an abstract business-oriented layer on the top of more traditional information movement mechanisms. This also provides process automation features, e.g., a view of how business information flows between different applications. This does not typically deal with physical integration flows and physical systems, but with abstract and shared processes.

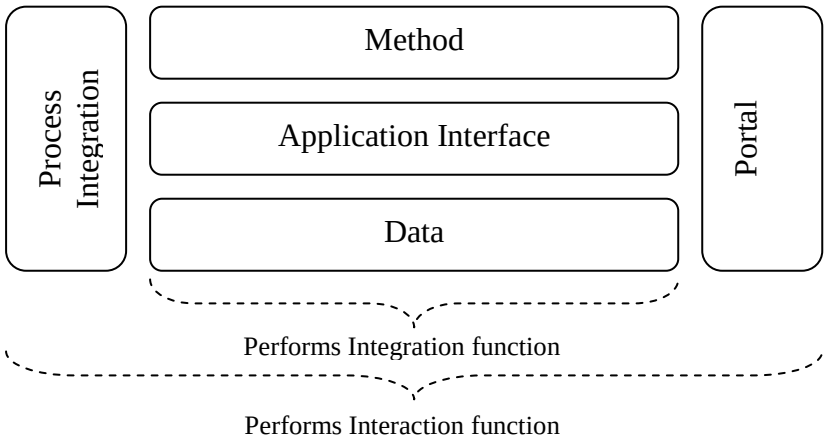


Figure 2.2 Application integration approaches

2.3.2 Middleware

Middleware is a simple mechanism that allows one entity (application or database) to communicate with another entity or entities. In other words, middleware is any

type of software that facilitates communications between two or more software systems. It is able to hide the complexities of the source and target systems, freeing developers from focusing on low-level application programming interfaces (APIs) and network protocols, allowing them to concentrate on sharing information [Kramp & Coulson 2000].

It is noted that middleware are evolving, which means that they may extend their functions beyond their originally designed functions. In the following, some well-known middleware types are discussed [Linthicum 2001]:

- ***Remote procedure call:*** Remote procedure calls provide developers with the ability to invoke a function within one program and have that function execute within another program on a remote machine. The fact that it is actually being carried out on a remote computer is hidden. Remote procedure calls require more bandwidth than other types of middleware products because carrying out a remote procedure call requires so much "overhead."
- ***Message-oriented middleware:*** This is a queuing software, using messages, which are byte sized units of information that move between applications, as a mechanism to move information from point to point. Because message-oriented middleware uses the notion of messages to communicate between applications, direct coupling with the middleware mechanism and the application is not required. Message-oriented middleware products rely on an asynchronous paradigm. Message-oriented middleware typically provides a structure (a schema) and content (data) in accord with the schema and its use of messages is relatively easy to manage.
- ***Distributed objects:*** Distributed objects are small application programs that utilize standard interfaces and protocols to communicate with one another. Two types of distributed objects are very popular today: common object request broker architecture (CORBA™) and component object model (COM™). CORBA™ and COM™ provide specifications that outline the rules that developers should follow when creating a CORBA™-compliant or COM™-enabled distributed object. CORBA™ is heterogeneous, with CORBA™-compliant distributed objects

available on most platforms. COM™ must be considered native to Windows ™ operating environments and therefore homogenous.

- ***Database-oriented middleware:*** This facilitates communications with a database, whether from an application or between databases. Database-oriented middleware are of two basic types: command line interfaces and native database middleware. An example of command line interface is Microsoft's open database connectivity (ODBC™). It exposes a single interface in order to facilitate access to different databases and uses drivers to accommodate differences between databases. Native database middleware accesses the features and functions of a particular database, using only native mechanisms.
- ***Transaction-oriented middleware:*** Transaction middleware does a commendable job of coordinating information movement and method sharing between many different resources. Although it provides an excellent mechanism for method sharing, it is not as effective at simple information sharing.
  - ***Transaction Processing monitors:*** They are based on the concept of a transaction, a unit of work with a beginning and end. The reasoning is that if the application logic is encapsulated within a transaction, then the transaction is either completed or rolled back completely. The load-balancing mechanisms of transaction processing monitors guarantee that no single process takes on an excessive load.
  - ***Application servers:*** They provide not only for the sharing and processing of application logic but also for connecting to back-end resources including databases, ERP applications, and even traditional mainframe applications. They also provide user interface development mechanisms and mechanisms to deploy the application to the platform of the web.
- ***Message broker:*** This facilitates information movement between two or more resources and can account for differences in application semantics and platforms. They can transform the schema and content of the information as it flows between various applications and databases.

The following table compares the different types of middleware available.

Table 2.4 Comparison of different types of middleware

	Remote Procedure Call	**Message-oriented middleware**	**Distributed Object**	**Database Oriented**	**Transaction Processing Monitors**	**Application Servers**	**Message Brokers**
Intruding	Does not change source and target application	Does not change source and target application	Change source and target application	Does not interfere with application logic	Changes the source code of the target application	Changes the source code of the target application	Does not interfere with the application logic
Connection mode	Point-to-Point	Point-to-Point and message queuing	Many-to-Many	Many-to-Many	Many-to-Many	Many-to-Many	Many-to-Many
Application Logic	Does not house logic	Does not house logic	Does not house logic, but allows for method-sharing	Does not house the whole application logic, but has the constraints posed by the application logic on the data	House application Logic	House application Logic	House application Logic
Degree of complexity	Simple	Simple	Complex	Fairly Complex	Complex	Complex	Complex
Flexibility	Not Flexible	Not Flexible	Not Flexible	Flexible	Flexible	Flexible	Very Flexible
Communication Mechanism	Synchronous (Direct Communication)	Asynchronous (Queued Communication)	Synchronous	Synchronous or Asynchronous	Synchronous	Asynchronous, Request/ Response	Asynchronous, Request/ Response

Table 2.4 Comparison of different types of middleware

	Remote Procedure Call	**Message-oriented middleware**	**Distributed Object**	**Database Oriented**	**Transaction Processing Monitors**	**Application Servers**	**Message Brokers**
Scalability	Do not scale well	Do not scale well	Do not scale well	Scale well	Scale well	Scale well	Excellent scalability
Supported Application Integration Approaches	Application Interface oriented	Application Interface oriented	Application Interface and Method Oriented	Database oriented	Application-Interface and Method Oriented	Application Interface, Method, Portal, Process integration oriented	Application Interface, Method, Portal, Process integration oriented
Dis-advantages	- Blocking middleware - It eats bandwidth	Data change usually needs adapters	The application must be created using distributed objects	The application logic is inseparable from the DB	Not so effective in information sharing	Not so effective in information sharing	Need separate middleware for each vertical industry
Special Features	Easiest to understand and use	Direct coupling between the applications is not required	Also a mechanism for application development and it provides 'the rules of the road' for developers	It doesn't interfere with the application and exposes a single interface to access different databases	Load balancing mechanisms	Not only shares logic, but also connects to back-end resources	Applications are cohesively connected and need not be changed for their interaction
Example	Distributed Computing Environment	Message Queuing Series	CORBA™, COM™	ODBC™, JDBC™	Tuxedo™	Oracle 9i™, WebLogic™	CentrPort™, Helio™

JDBC- Java Database Connectivity
ODBC-Open Database Connectivity
COM-Component Object Model
CORBA- Common Object Request Broker Architecture

2.3.3 Adapters

Adapters play a key role in application integration. There are two different types of adapters existing on the market today. The first type of adapter is more or less like a connector. It is a basic communication interface into or out of a particular system or database. It also connects the application to the middleware products. In this case, only the middleware has the application logic in it to integrate different applications. The second type of adapter has the application logic in it to integrate different applications. Consequently, the second type acts more or less like middleware. Both these types of adapters are explained in detail below.

- ***Adapters with application logic:*** These adapters deliver all-important business logic at the application sub-module level. Any adapter without application logic, combined with many lines of hard code, can get data from SAP™ manufacturing into an Oracle™ financial application, for instance. In contrast, an intelligent adapter should be able to insert the data into the proper accounts payable field or invoice line item and meet any pre-determined business requirements when doing so. This requires application-level logic to perform the necessary actions [Traverse 2001]. The best adapter solutions include an intuitive, template-based adapter development environment. They reduce the need for low-level programming and let customers build or modify adapters in a graphical, drag-and-drop environment.
- ***Adapters without application logic:*** This type of adapter acts just like a connector. The middleware has the knowledge about where the data would be and what it means semantically. The adapter simply converts the data from its native format to the format compatible with the middleware. Some of the adapters of this type are called “Universal Adapters.” They provide middleware-independent transformation and universal adapter development tools. Figure 2.3 shows the architecture of a universal adapter [eJai 2001].

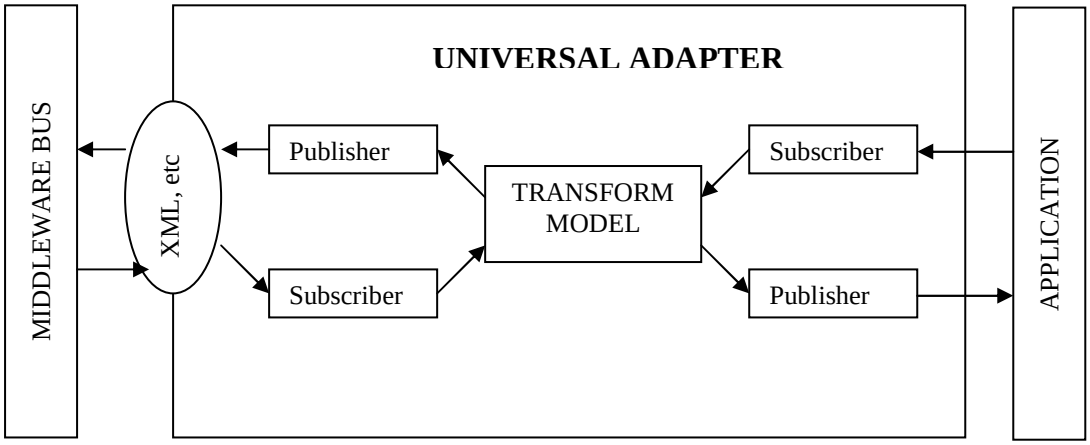


Figure 2.3 Universal adapter

Universal adapters have transform models inside them that convert data from the native application format to the format required by the middleware. They handle publishing and subscribing in addition to reply/request functions to enable application formats usable by middleware [eJai 2001].

2.4 Further Notes on Commercial Tools for Application Integration

In this section, some of the popular tools available on the market that can be useful to application integration are listed. Some of these were discussed in previous sections under specific contents. Here, they will be discussed with focus on their purposes/roles, ways of achieving their goals, assumptions made during their development, their weaknesses, and possible future developments.

2.4.1 Common Object Request Broker Architecture (CORBA™)

CORBA™ was developed by the Object Management Group. It is an open, vendor-independent architecture and infrastructure that computer applications use to work

together over networks. The core process in CORBA™ is an object request broker (**ORB**).

Using the standard protocol **IIOP** (Internet Inter-ORB Protocol), a CORBA™-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA™-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

CORBA™ applications are composed of objects, individual units of running software that combine functionality and data, and that frequently (but not always) represent something in the real world. In the case of legacy applications, they are wrapped in code with CORBA™ interfaces and opened up to clients on the network. The separation of interface from implementation, enabled by interface definition language, is the essence of how CORBA™ enables interoperability. For each object type an interface is defined in interface definition language. Any client that wants to invoke an operation on the object must use this interface definition language interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the same interface definition is used there to unmarshal the arguments so that the object can perform the requested operation with them. The interface definition is then used to marshal the results for their trip back, and to unmarshal them when they reach their destination. In the case of remote invocation, the ORB examines the object reference and discovers that the target object is remote. Then it routes the invocation out over the network to the remote object's ORB.

CORBA™ has assumptions at two key levels: first, it assumes that the client knows the type of object it is invoking and that the client and object interfaces are generated from the same interface definition language. This means that the client knows exactly which operations it may invoke, what the input parameters are, and where they have to go in the invocation; when the invocation reaches the target,

everything is there and in the right place. Second, the client's ORB and object's ORB are assumed to agree on a common protocol - that is, a representation to specify the target object, operation, all parameters (input and output) of every type that they may use, and how all of this is represented over the wire.

In CORBA™, programs are made to treat all objects as (potentially) remote, which may result in an awkward coding style. CORBA™'s assumptions about object naming are not scalable. This is a problem for very large object-based applications because some of the objects may contain millions of other objects. Security schemes that prevent unauthorized users from gaining access to a computer network, firewalls, can be deployed with full effectiveness only within an intranet. CORBA™ can be implemented to full effect in UNIX™ only. Cross-platform porting is very difficult. All major vendors have left the CORBA™ consortium. CORBA™ is now only a niche market supported by small companies.

The next generation of CORBA™ is needed. Simple object access protocol (**SOAP**) is the primary contender for that. SOAP is fully inspired by CORBA™ and several of the CORBA™ designers have designed SOAP. The IIOP and IIOP binary data in CORBA™ are replaced by **HTTP** (Hyper Text Transfer Protocol) and **XML** (Extensible Markup Language) in SOAP.

2.4.2 Java 2 Platform, Enterprise Edition (J2EE™)

J2EE™ defines a standard for developing multi-tier enterprise applications. It simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming. Java 2 Enterprise Edition™ adds full support for Enterprise JavaBeans™ (**EJB**) components, Java servlets API™, JavaServer Pages™, and XML technology.

The enterprise java beans (EJB™) specification provides one type of interoperability between applications or between applications and databases. They are designed to give middleware developers a standard specification for supporting Java™ server components. EJBs™ created with different tools can, at least in theory, interoperate with each other and can run on any application server that supports the EJB™ specification. Another new specification, expected to appear in the next rendition of the Java 2 Platform Enterprise Edition (**J2EE**™), is the Java Connector Architecture™. The Java Connector Architecture™, currently under development, would define a common architecture for connecting EJB™ servers to heterogeneous enterprise systems such as ERP, mainframe transaction processing, and database systems.

The J2EE™ standard wraps and embraces existing resources required by multi-tier applications with a unified, component-based application model. This enables the next generation of applications for solving the strategic requirements of the enterprise. Remote method invocation is a feature of Java™ that facilitates inter-application communication. When EJBs™ or other Java™ objects need to communicate over a network with other Java™ applications, they use remote method invocation, a remote procedure call that enables one Java™ application to access the objects and methods of another Java™ program across a network. For communication between Java™ and non-Java™ applications, a new specification released in 2001 called remote method invocation over **IIOP** (Internet Inter-ORB Protocol), allows Java™ remote method invocation objects to communicate with non-Java™ CORBA™ objects over a network. Instead of using the Java™ remote method protocol for interactions between distributed objects, which is what the remote method invocation employs, remote method invocation over IIOP uses IIOP, a protocol originally developed to connect CORBA™ products from different vendors. The Java Connector Architecture™ is to be based on existing Java™ technologies, including Java Transaction Application Program Interface™, EJBs™, and Java Database Connectivity™ (**JDBC**).

Entity EJBs™ are developed to follow loosely the relational data model: basically one instance of the particular entity bean class equals one row in the relational table. J2EE™ assumes that all the enterprise application can be built with the Java™ technology, either EJB™ or Java Connector architecture™.

The overall complexity of the system is increased with the introduction of EJBs™. The issue of data security and user access is also more complicated because while Sun's EJB™ specification offers its own role-based authentication scheme, its actual implementation is left to the EJB™ vendor.

The focus of vendors is slowly moving away from the J2EE™ server and into complementary products that will be built as J2EE™ applications, running on top of their existing J2EE™ servers. These new applications will leverage built-in support for transaction handling and scalability provided by J2EE™ application servers, providing a fully integrated platform for development. ColdFusion™ is one such exciting example. Macromedia™/Allaire™ plans to port the ColdFusion™ development platform to a generic J2EE™ application, which can run on top of any J2EE™ compliant application server.

2.4.3 Microsoft BizTalk Server™

The BizTalk™ server is a middleware product used for application integration. It is a collection of tools and services that helps the enterprise activities in a variety of ways. The BizTalk™ server ships with BizTalk™ editor, BizTalk™ mapper, BizTalk™ document tracker, BizTalk™ orchestration designer, BizTalk™ messaging manager, and BizTalk™ server administrator. The BizTalk™ editor creates document specifications that represent extensible markup language (**XML**), flat files, and electronic data interchange files. The BizTalk™ editor also imports existing document type definition and XML-data reduced schema. Mapping document contents between different formats is done by the BizTalk™ mapper. The BizTalk™ document tracker, the BizTalk™ orchestration designer, the BizTalk™ messaging

manager, and the BizTalk™ server administrator performs the functions of tracking and auditing messages, building business processes, managing trading relationships, and managing BizTalk™ server, respectively. The BizTalk™ server gives industry-leading support for standards such as XML, simple object access protocol (**SOAP**), secure/multipurpose Internet mail extensions, and public key infrastructure. The BizTalk™ server also has accelerators and adapters.

The purpose of the BizTalk™ server is to easily integrate the internal applications, securely connect with the business partners over the Internet, and rapidly automate the business processes. This infrastructure enables companies to integrate, manage, and automate business processes by exchanging business documents (e.g., purchase orders, invoices) among applications within or across organizational boundaries. The BizTalk™ server makes the enterprise application integration simple by integrating with virtually any product or technology.

There are several approaches for integrating the existing legacy applications. If the existing application can output a file format that BizTalk™ server can read, then the Biztalk™ server acts as simple middleware and integrates the two applications. A more automated approach would be to write an application adapter using the documented support from the BizTalk™ to feed data directly into and out of a legacy application. BizTalk™ server orchestration is another way of integrating applications. It is an environment for creating and running distributed business processes. Within this environment, a business process can directly access a web service as an activity in that process using the SOAP Toolkit. SOAP is the envelope format for BizTalk™ framework documents and provides a reliable messaging protocol for the BizTalk™ server. The BizTalk™ server has a comprehensive library of over 300 adapters, which ensures the easy integration of products and technologies with the BizTalk™ server. BizTalk™ accelerators include a powerful combination of product enhancements, simple-to-use tools, documentation, and samples that are developed in concert to ensure they work well together.

The BizTalk™ server works with the assumption that adapters can be implemented upon all the legacy applications that need to be integrated. It also assumes that there exists a standard format through which the legacy applications that need to be integrated can talk to each other.

It also has some weaknesses or disadvantages. Some of them are: it does not integrate as well with non-Microsoft™ objects; it needs partners to access legacy data; it does not address XML programming barriers (addresses only standard document object model); it does not work with UNIX™.

BizTalk™ server has an open, extensible environment. Custom transports and application integration components can be developed to access legacy systems. Custom parsers, serializers, and correlators can be developed to handle file formats not supported in the core product. Custom "functoids" can be developed to extend the mapping capabilities of the BizTalk™ server. Custom schema importers can be developed to extend the BizTalk™ server editing tool.

2.4.4 Adapter with application logic

An adapter is an application integration device and adapts the source application format in a way that it can be integrated with the target application format. It uses the intelligence it contains regarding the business logic to integrate the source and the target application.

In the case of integrating a legacy application on the mainframe with a client application, the application integration adapter allows the legacy applications to remain untouched while providing an interface to clients. An application integration adapter has the intelligence to identify the host application and to provide message reformat, security and data connectivity for the appropriate application – all without any change being made to the existing legacy mainframe application.

An application integration adapter routes the transaction to the legacy mainframe systems via the proprietary communications environment of the mainframe system. On the legacy mainframe system, the destination application will receive the routed transaction from the application integration adapter. The application processes the transaction and returns a reply to the application integration adapter via the proprietary communications protocol, where the message is reformatted and returned to the originating workstation via the industry standard transmission control protocol/Internet protocol (**TCP/IP**).

The important assumption by the application integration adapter is that all the application formats in use today can be converted into a standard format. Another assumption by the adapter is that the whole of the application logic is deterministic and can be contained in the adapter.

The main disadvantage of the adapter is that it is not possible to convert all the formats an application may have into a standard format. Containing the whole application logic in the adapter is a complex process. Even if one does so, problems will be created when the application logic changes.

The types of adapters available on the market are ready for evolving into universal adapters with application logic. They must be able to have a transformation engine for all possible formats of an application.

2.5 General Concluding Remarks and Discussions

Based on the review of two categories of solutions to enterprise application integration (EAI), it may be clear that the first category of solutions, i.e., those which stemmed from many computer software giants such as Microsoft™ and Sun™, has attempted to provide generic and fundamental methods for different applications for communicating within both Intranet and Internet. Though they have

achieved their goal in a specific period of time, their life cycles vary; some fade away very soon, while others may stay with enterprise applications longer. This evolutionary nature has actually produced a huge difficulty for enterprises, which need to make decisions regarding their investment on information technology for their business processes. Apparently, the first category of solutions has not provided complete solutions to the EAI problem.

The second category of solutions reflects the efforts made in manufacturing or in the industrial engineering society. The most useful point here is the need to have enterprise ontology for EAI. However, existing studies on enterprise ontology modeling have not included ontology or data modeling for enterprise applications. Knowledge of enterprise applications relevant to EAI does exist but is represented in a way specific or proprietary to individual producers of them. Fundamental representation (i.e., ontology) of the enterprise applications is needed for the purpose of integrating them in a particular enterprise environment.

Further research towards the following directions is needed based on the analysis above. *Firs*t, enterprise ontology modeling should include enterprise functions and enterprise applications, as well as their connections. *Second*, a more general modeling language should be used for practicability and usability. *Third*, the relationship between the complete enterprise model and the various solutions from the first category of solutions (e.g., adapters, middleware, etc.) should be such that they are treated as individual connection tools organized in a knowledge base, which may be called the tool knowledge base, and they will be called or assigned by a high level decision engine to enable communication at the enterprise application level.

CHAPTER 3

UNIFIED MODELING LANGUAGE

3.1 Introduction

Unified modeling language (UML) was developed by Grandy Booch, James Rumbaugh, and Ivar Jacobson in 1996 and later standardized by the Object Management Group in 1997. UML is one of the most powerful tools for modeling a dynamic system. The purpose of this chapter is to explain how UML fulfills its modeling promises through a set of notions/concepts and constructs. For this purpose, this chapter will be organized as follows. In Section 3.2, some of the basic notions of UML are illustrated. In Section 3.3, a powerful mechanism in UML for users to extend the UML constructs is described. In Section 3.4, a textual formalism of UML, called object constraint language (OCL), is introduced. Section 3.5 focuses on a discussion of database design using UML. There is also an analysis of the advantages of the UML approach for database design in Section 3.6.

3.2 UML Basics

UML is a very expressive language, addressing all the views needed to develop and then deploy systems ranging from enterprise information systems to distributed web-based applications and even to hard real time embedded systems. UML is used

to visualize, construct, and document the artefacts of a software intensive system [Booch et al. 1999]. UML has a notation and a well-defined set of syntactic and semantic rules [Eriksson & Penker 2000]. The strongest aspect of UML is its incorporation of views. UML takes into account five types of perspectives, or views, pertaining to any given piece of software. In the following, these views, along with the UML graphical notation are introduced.

3.2.1 Functional view

The functional view represents how the user will view the software in terms of its functions. This view is constructed by **use-case diagrams**, which show the over-all functionality of software. It ignores how the software goes about performing its task and focuses on what is being performed. In Figure 3.1, the scenario where a student is enrolled in a university is considered. The **Student** and the **International Student** are users and are represented diagrammatically as shown in Figure 3.1. The International Student 'is-a' Student, and this generalization association is represented by an arrow with a hollow triangle end, originating from the more specialized user, International Student in this case. **Enrol Student** and **Enrol International Student** are use-cases, which are performed for the users. The use-cases are represented diagrammatically as an oval shown in Figure 3.1. Enrol Student is extended into Enrol International Student and this extension relationship is represented by an arrow and a stereotype (see the next section) **<<extend>>**.

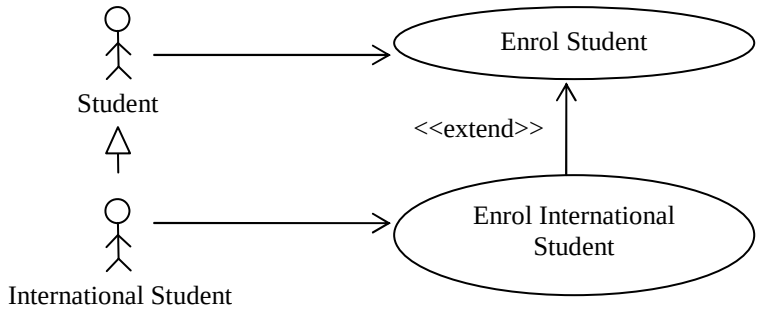


Figure 3.1 Use-case diagram for enrolling students in the University

3.2.2 Structural view

A system is composed of classes, objects, and their associations. The structural view represents the system from these perspectives. This view does not display how the classes and objects actually behave, but shows their relationships. A structural view can be constructed with two types of diagrams, class and object diagrams.

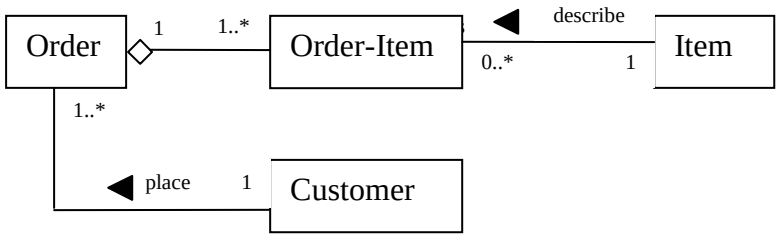


Figure 3.2 Class diagram for modeling an order

Class diagrams: They describe the structure of a system. The structures are built from classes and relationships. The classes can represent and structure information, such as products, documents, and organizations. In Figure 3.2, a typical customer order is modeled using the class diagram. The classes are represented as rectangles with the names of the class written inside the rectangle. In Figure 3.2, **Order**, **Order-Item**, **Item**, and **Customer** are all classes. The association name appears near the association line, and the multiplicities appear on each end. In Figure 3.2, **describe** and **place** are association names, and the numbers are the multiplicities. The customer places an order, which is shown by an arrow pointing towards order. The aggregate relationship is a specialization of association, where a whole is associated with its parts. The hollow diamond on the association line in Figure 3.2 denotes that an order aggregates many order-items. The multiplicity of the association can be represented as either a range (1..*), (0..*), etc., or a specific number (1). In the association between customer and order, the multiplicity icons represent the scenario where one customer places one or more orders. The semantics of the order-item and item classes with associations is that (i) all items

and their definitions/descriptions are gathered with the Item class, (ii) not all items in the Item class are put on orders by customers.

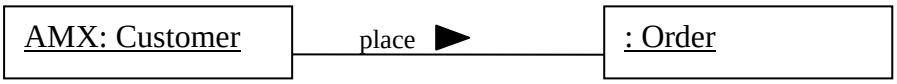


Figure 3.3 Object diagram for modeling a specific customer order

Object diagrams: They express possible object combinations of a specific class diagram. They are typically used to exemplify a class diagram. Because an object diagram is an instance of a class diagram, multiplicity is not shown. The object names, which are underlined, are composed of their names, followed by a colon and the class name. In Figure 3.3, **AMX** object belongs to the class **Customer**. As UML is extendible, the object instantiated from the class is represented in this thesis as shown in Figure 3.4.

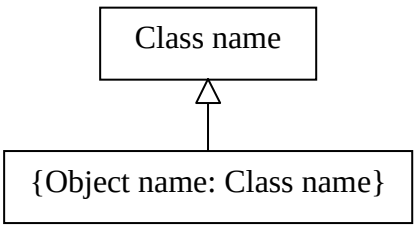


Figure 3.4 Object instantiated from class

3.2.3 Behavioral view

This view represents the actual behavior of software. It contains diagrams representing the inner-workings of classes and their behaviors in respect to one another. The behavioral view is composed of four diagram types.

Sequence diagrams: They show one or several sequences of messages sent among a set of objects. The inter-class communication is broken down as a series of steps. Messages are shown as arrows that represent communication between objects. They

follow similar association rules as mentioned in the earlier views. Lifelines are vertical dashed lines that indicate the object's presence over time. Activation boxes represent the time an object needs to complete a task and are represented by rectangular boxes along the lifelines. Objects can be terminated early using an arrow labelled **<<destroy>>** that points to an X. All these concepts are represented in Figure 3.5 with an example.

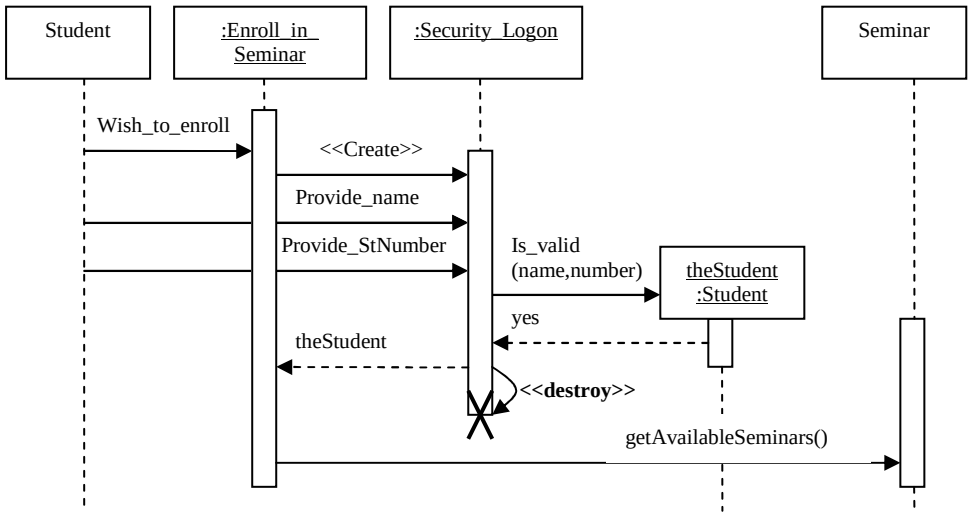


Figure 3.5 Sequence diagram for enrolling a student in the seminar

Collaboration diagrams: They describe a complete collaboration among a set of objects with their roles. Unlike sequence diagrams, collaboration diagrams do not have an explicit way to denote time; instead the messages are numbered in the order of execution. Sequence numbering can become nested, for example, nested messages under the first message are labelled 1.1, 1.2, 1.3, and so on. In Figure 3.6, the sequences of messages passed between different objects are represented.

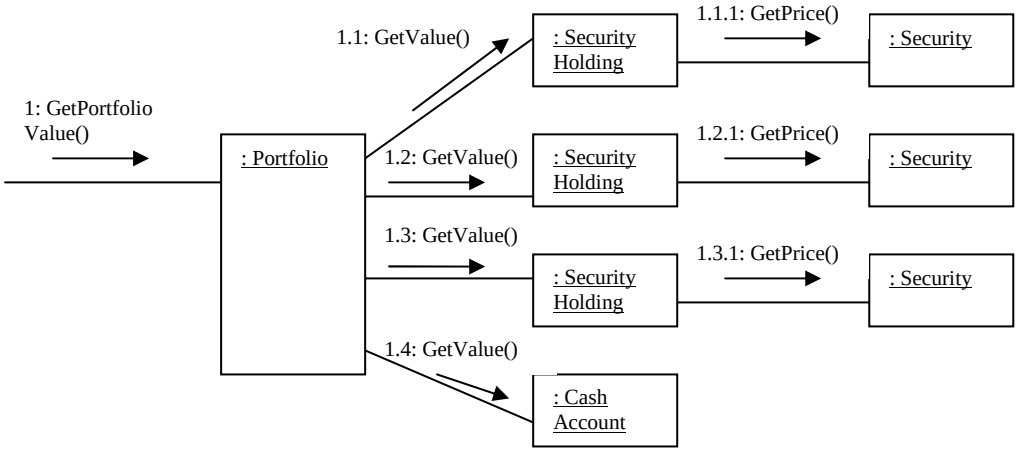


Figure 3.6 Collaboration diagram showing the calculation of the value of a portfolio

Statechart diagrams: They express object state under differing circumstances. External stimuli are the focus of statechart diagrams. States represent situations during the life of an object and can be illustrated using a rectangle with rounded corners. Changes in states are indicated with an arrow pointing from one state to another. The state transition is labelled with its cause. A filled circle followed by an arrow represents the object's initial state. An arrow pointing to a filled circle nested inside another circle represents the object's final state. The above-mentioned concepts are represented in Figure 3.7, which is a statechart diagram for invoices.

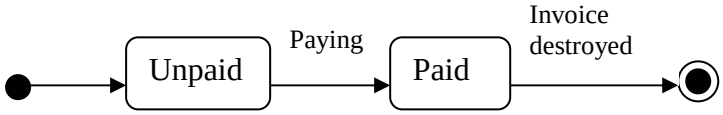


Figure 3.7 Statechart diagram for invoices

Activity diagrams: they are much like statechart diagrams in that they express the response of classes with respect to execution, but activity diagrams focus on variables and states internal to a system rather than external stimuli. Activities are action states represented by a rectangle with rounded corners. The receiving activity is represented as a rectangle with a concave side and the sending activity as a

rectangle with a convex side. Action flow arrows illustrate the relationships among activities. The initial and final states are represented as mentioned in the statechart diagrams section. A diamond represents a decision with alternate paths. In Figure 3.8, the two alternate decision paths, i.e., to **send rejection** or to **send acceptance**, are represented by a diamond. The outgoing alternates should be labelled with a condition or guard expression. A synchronization bar helps illustrate parallel transitions. In Figure 3.8, either the **send rejection** or the **receive invoice** activity could cause a final state in the activity.

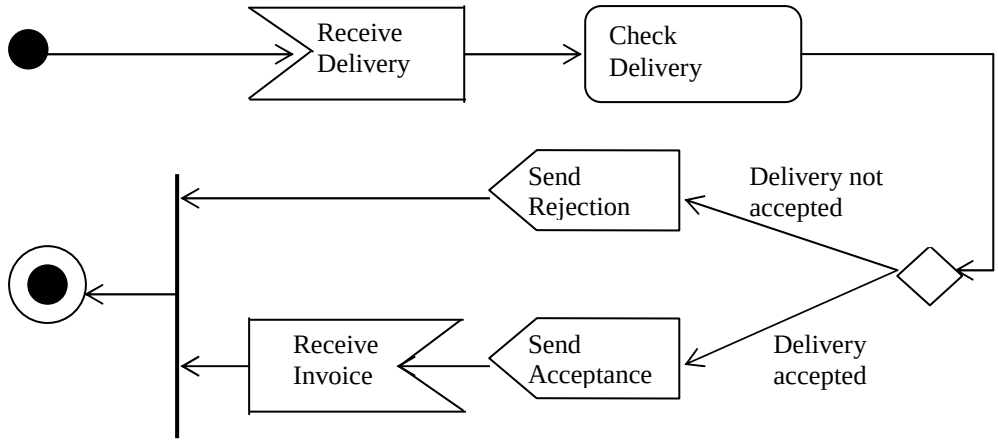


Figure 3.8 Activity diagram for receiving delivery

3.2.4 Implementation view

This view demonstrates the organization and dependencies of the actual system and its pieces. This view has only one type of diagram.

Component diagrams: Show the organization and dependencies of a system's actual parts. The components are represented using rectangles with tabs and their relationships using dashed arrows as shown in Figure 3.9.

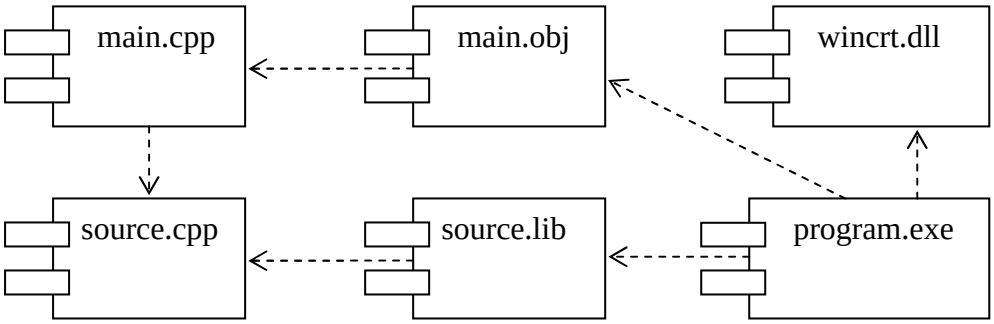


Figure 3.9 Component diagram showing dependencies between software components

3.2.5 Environment view

This view represents the environment in which the software will exist when finished. This view has one diagram type.

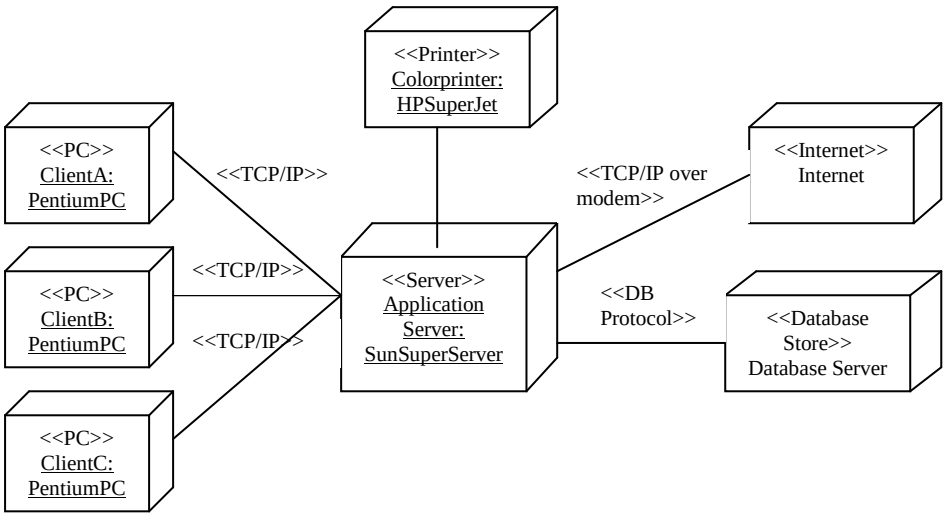


Figure 3.10 Deployment diagram of physical hardware in the system

Deployment diagrams: Show all required resources for the software. This is a special case of class diagram used to describe the hardware within a software

system. The nodes are represented by cubes, and their associations by lines, as shown in Figure 3.10.

3.3 Extending UML

UML provides certain constructs that are used to customize the building blocks mentioned in the previous section to suit application needs. Because of UML's unique capability to adapt and extend, it is possible to add new building blocks to UML, making it a very flexible language that can be used in many situations [Eriksson & Penker 2000]. These constructs are:

- *Stereotypes:* These are techniques used to define new kinds of building blocks in UML based on existing blocks. They allow one to customize an existing UML modeling element to suit application requirements.
- *Tagged values:* All modeling elements can be extended with tagged values that consists of a tag and a value, for example, a version tag and a version number (value) for a class.
- *Constraints:* These are rules applied to UML models. They can be applied to just one or to several model elements. In addition to using predefined constraints, such as 'xor' and 'ordered', customized constraints, can be defined through a textual language of UML called object constraint language (OCL) (to be discussed in Section 3.4).

To facilitate the use of UML for modeling business processes, Eriksson & Penker [2000] developed several more applicable model templates, which are discussed below.

- *Processes:* The Eriksson-Penker business extensions represent a process in a UML class diagram with the process symbol shown in Figure 3.11. A process (an activity stereotyped to process) takes input resources from its left-hand side and indicates its output resources on its right-hand side. The goals of the process are

illustrated as goal objects above the process symbol and resources used as part of the process are represented as resource objects below the process symbol, as shown in Figure 3.11.

- *Resources:* Resource types are represented as classes. Resource instances are represented as objects. The Eriksson-Penker business extensions define some stereotypes to indicate different categories of resource types. Physical, abstract, information object, and people are four types of resources. The information resource class alone is represented by a rectangle with slanting sides.
- *Goals:* A goal describes the desired state of one or more resources. Dividing goals into sub-goals is called goal modeling. The Eriksson-Penker business extensions represent goals as objects and use an object diagram to show the dependencies between goals and sub-goals.
- *Rules:* Many of the UML diagrams have built-in support for defining rules, using the generic construct of defining a constraint. A class diagram has structural constraints in its relationships (e.g., the multiplicity of an association). A state diagram has behavioral constraints in its state and state transitions. Derivation rules can be defined as a computational constraint in UML using object constraint language (see Section 3.4).

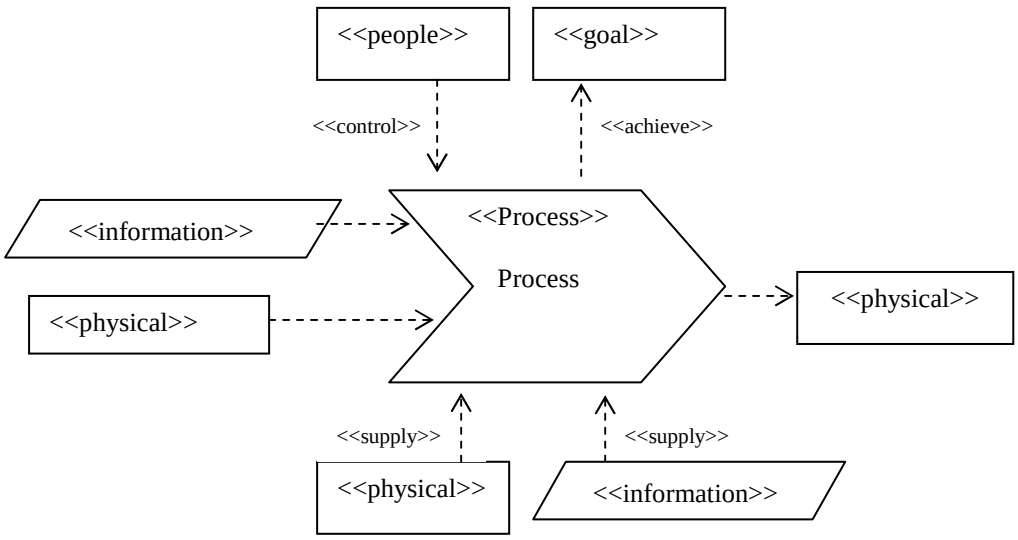


Figure 3.11 Generic process diagram

3.4 Object Constraint Language (OCL)

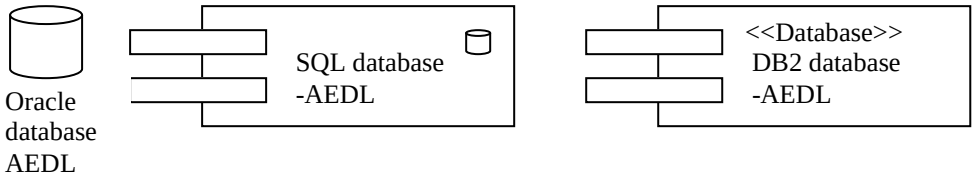
Object constraint language (OCL) is UML's recommended language for specifying constraints. OCL is a declarative specification language. Statements in OCL cannot actually change anything in the model, but they can however specify such a change. The OCL language consists of expressions, comprising a statement involving operators and operands, which return a result value. All expressions are related to a specific context, which is specified with the keyword '*context*' followed by the name of the context part. To represent the concept that the OCL constraint is a class invariant on the context, the keyword '*inv:*' followed by the invariant expression is used. This means that the expression must evaluate to true for all objects of that class. OCL will be used in Chapter 4 to model an enterprise.

3.5 Application of UML for Data Modeling

3.5.1 UML data modeling profile

The data modeling profile for UML explains the stereotypes used in UML for database specific needs [Rational 2000]. It includes the descriptions and examples for important concepts including database, schema, table, key and relationship. The profile uses stereotypes and tagged values for all the information needed to describe the structures of the database and its elements. The profile also uses constraints to enforce database design conformance [Naiburg & Maksimchuk 2001].

- *Database:* A database is a system for storage and controlled access to stored data. It is the biggest element a data model supports. The stereotype <<Database>>, when used as a UML component, defines a database. As a component, the database must have a name. The three possible representations of a database component are shown in Figure 3.12.



AEDL: Name of the database

Figure 3.12 Database representation using UML

- *Schema:* A full description of the data model to be used for storage and retrieval of data is stored in a schema inside a database. The schema is the biggest unit that can be worked with at any given time. A package (a rectangle with a tab on top, as shown in Figure 3.13) using the <<Schema>> stereotype in a UML model represents a database schema. Figure 3.13 represents the schema.

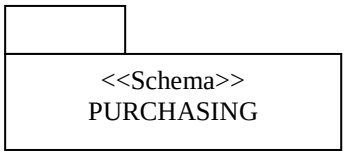


Figure 3.13 Schema representation using UML

- *Table:* A table is the basic modeling structure of a relational database. It represents a set of records of the same structure, also called rows. Each of these records contains data. The information about the structure of a table is stored in the database itself. A class with the <<Table>> stereotype represents a relational table in a schema of a database. The table can be represented in different ways, as shown in Figure 3.14. The first compartment represents the table name; the second compartment represents the attributes or column names of the table; and the third compartment represents the operations of the table.

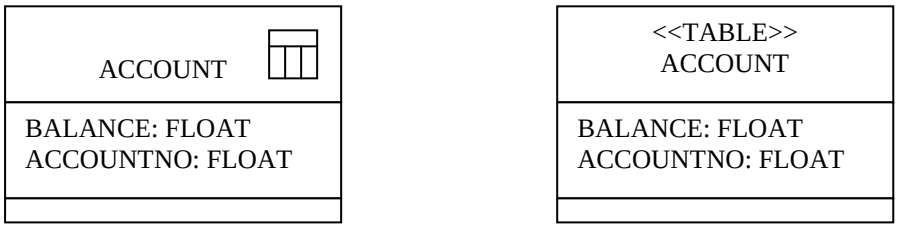


Figure 3.14 Table representation using UML

- *Key:* Keys are used to access a table. Primary keys uniquely identify a row in a table, while foreign keys access data in other related tables. In Figure 3.15, the PK tag represents the primary key, and the PK stereotyped operation is the primary key constraint. Similarly, an FK tag represents the foreign key and it generates the foreign key constraint, which is represented by a stereotyped FK on the operation.

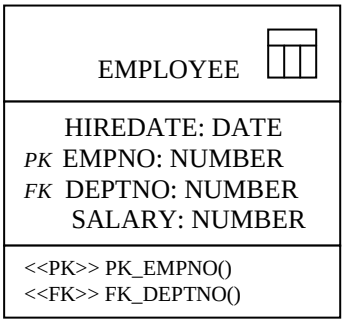


Figure 3.15 Representation of keys using UML

- *Relationship:* A dependency of any kind between tables in a data model is called a relationship. Every relationship is between a parent and child table, where a parent table must have a primary key defined. The child table creates a foreign key column and foreign key constraint to address the parent table. A non-identifying association represents a relationship between two independent tables, whereas an identifying relationship is a relation between two dependent tables. In Figure 3.16, DEPT and EMP are the parent and child tables respectively, where the foreign key of the child table does not contain all of the primary key columns. In Figure 3.17, PERSON and ACCOUNT are the parent and child tables

respectively, where the idea that the child table cannot exist without the parent table is represented by the containment relationship (filled diamond).

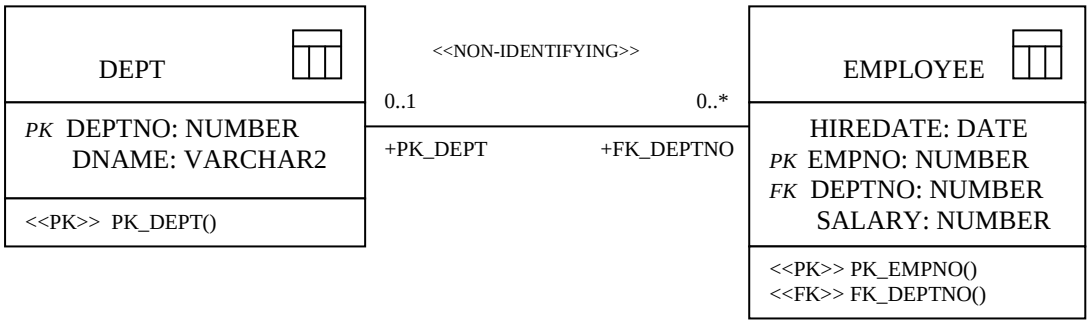


Figure 3.16 Representation of non-identifying relationship using UML

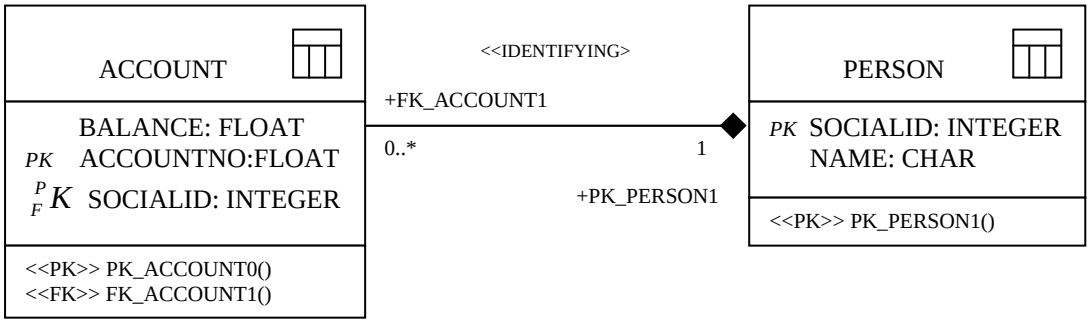


Figure 3.17 Representation of identifying relationship using UML

A relationship has two roles associated with it. They define the role of one table in association with the other. It is possible to assign more than one relationship between two tables using different roles.

- *Datatype:* Supporting relational databases requires the support of standard datatypes. Examples are char, date, float, long, and number.
- *Column:* A table contains columns, which are tagged attributes. Columns, when they are instantiated as a row, can contain data. They must have a defined data type. Constraints can be checked for any column. Columns are modeled as attributes of the table class.

- *Constraint:* A constraint is a rule applied to the structure of the database. This rule extends the structure and can be applied to a column and/ or table. All constraints are defined as stereotyped operations.
- *View:* A virtual table that, from the user's perspective, behaves exactly like a typical table but has no independent existence of its own. The view is modeled as a class, with the view icon drawn as a table with dotted line. It is also modeled as a class with stereotype <<View>>.
- *Role and cardinality:* This represents a numerical range defined on the relationship of how many times the relationship can occur. The role describes the relationship between two classes textually. The role and cardinality are mentioned when relating two classes.
- *Stored procedure:* This is an independent procedural function that typically executes on the server. It can be defined as a procedure within the database, from an external file or as a function. A stored procedure container is a grouping of one or multiple stored procedures. It is modeled as a class with the stereotype <<SP Container>>. Within a <<SP Container>> are the stored procedures, which are modeled as operations on the container with the stereotype <<SP>>.
- *Trigger:* A trigger is activity executed by the database management systems as a side effect or instead of a modification of a table or view to ensure consistent system behavior on data operations. The trigger is displayed as a trigger stereotype on the operation.
- *Index:* An index is a physical data structure that speeds up data access. It does not change the quality or the quantity of data retrieved. The index is represented as index stereotype on an operation.

3.6 Advantages of Using UML for Data Modeling

UML has many advantages over many other languages used for data modeling. They include:

- UML gives the ability to model, in a single language, the business application and the database architecture of systems. It is helpful in bringing the many teams involved in the development process together by using the same language (UML) to model both the applications and data.
- Modeling a database is generally just the modeling of database tables, columns, and relationships but not the entire database design. UML describes the database in great detail. All the aspects of the database can be modeled by using UML. Designing the database through data modeling in UML provides the ability to capture many more items on the diagram visually than with traditional entity-relationship (E-R) notations. One can model elements like domains, stored procedures, triggers, and constraints as well as the traditional tables, columns, and relationships. It exposes elements directly on the model that normally get hidden behind modeled elements as tagged values.
- In UML for data modeling, diagrams do not have to be typed; this means one can have elements of many different types on a diagram or one can stereotype the diagram to a specific type and allow only those types of elements. UML supports the typed concept for different levels of constructs. Making a database typed, such as <<Database Diagram>>, is a very powerful mechanism for allowing a whole database to be manipulatable. In data modeling, typed or non-typed construct has triggered a long-standing debate in database and artificial intelligence communities. In general, typed constructs tend to be more efficient in retrieval and processing of data in the case of a large amount of data of a few types, while non-typed constructs tend to be more flexible and suitable in the case of a small amount data but a variety of data types. The mechanism for combination of typed and non-typed constructs could achieve a more in terms of object relativity that has been one of the criteria when evaluating a data modeling method [Smith & Smith 1977, Ter Bekke 1992, Zhang & Werff 1994]. For example, a data structure called ‘Instance-As-Type’ [Li et al. 1999] could be modeled with this mechanism.

CHAPTER 4

ENTERPRISE SEMANTIC MODEL FRAMEWORK

4.1 Introduction

This chapter aims to introduce and describe the enterprise semantic model (**ESM**) framework. Section 4.2 describes the basic idea of the ESM framework, in which the philosophical ground of the semantic model approach to enterprise application integration (**EAI**) is elaborated further. In Section 4.3, the characteristics of the **ESM** framework are discussed, and a fundamental notion, ontology, is elaborated. This has led to the need of two types of ontology for **EAI**: ontology for enterprises and ontology for enterprise information systems. Sections 4.4 and 4.5 present the conceptual models of ontology for enterprises and of ontology for enterprise information systems, respectively. Section 4.6 provides a systematic procedure for developing an ontology for a particular type of enterprise, a process enterprise. In Section 4.7, the procedure for developing an ontology for a particular type of process enterprise, a steel enterprise, is described. In Section 4.8, the enterprise semantic model template is developed by using the ontologies developed in the previous sections.

4.2 Fundamentals of the Semantic Model Approach for EAI

4.2.1 Semantic data model

Historically, semantic database models were first developed to facilitate the design of database schemas. In the 1970s, traditional models (relational, hierarchical, and network) used data structures that focused on how they were represented in computers. These data models lacked direct support for relationships, data abstraction, inheritance, constraints, unstructured objects, and the dynamic properties of an application. Although the relational model has provided database practitioners with a modeling methodology independent of the details of the physical implementation, many database designers believe that the relational model does not offer a sufficiently rich conceptual model for problems that do not map onto tables in a straightforward fashion.

During the past decade numerous data models with the aims of providing increased expressiveness to the modeller and incorporating a richer set of semantics into the database have emerged. This collection of data models can be loosely categorized as “semantic” data models. Semantic models provide a higher level of abstraction for modeling data, allowing database designers to think of data in ways that correlate more directly to how data arise in the world. The primary components of semantic data modeling are the explicit representation of objects, attributes and relationships among objects; type constructors for building complex types; ‘is-a’ relationships; and derived schema components. Additionally, the semantically based data models provide the following advantages over traditional, record-oriented systems [Trujillo et al. 1997]:

- Increased separation of conceptual and physical components;
- Decreased semantic overloading of relationship types;
- Availability of convenient data abstraction mechanisms.

4.2.2 Semantic data model as a means of integrating program/data

The role of a semantic model can be re-visited. The semantic model in essence plays a role as an integrator (see Figure 4.1). Figure 4.1(a) shows the evolution from program pattern to semantic model pattern, and Figure 4.1(b) shows the architecture of information systems based on the semantic model concept. The first sense of this integrator is that different programs and files work together with the database concept; the second sense is that different databases/programs supported by different DBMSs work together based on the semantic model concept (see Figure 4.1(a)).

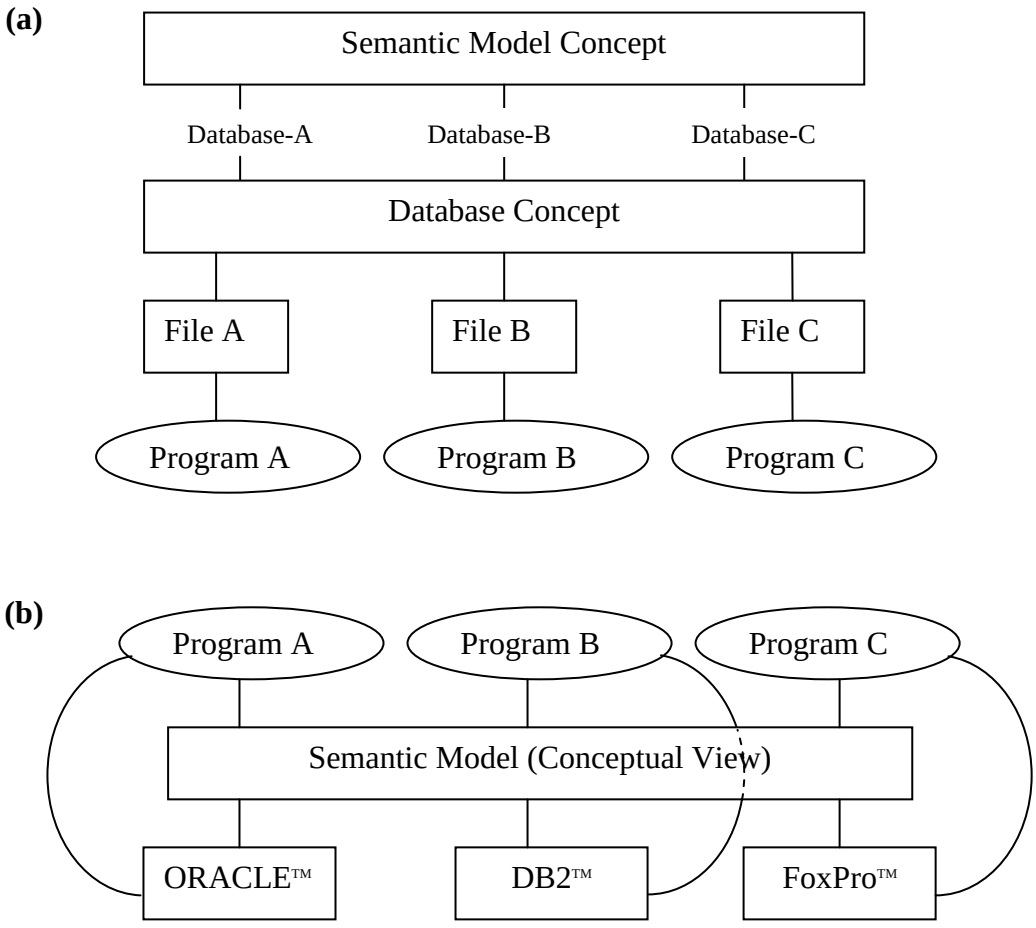


Figure 4.1 Semantic model as an integrator

Under the semantic model concept, programs (A,B,C) with their corresponding DBMSs work in the way shown in Figure 4.1(b). It is noted that the operation architecture, as shown in Figure 4.1(b), has implied the need of an information management system for the semantic model. At this point, research with the heading of database integration is along this path [Ma et al. 1999].

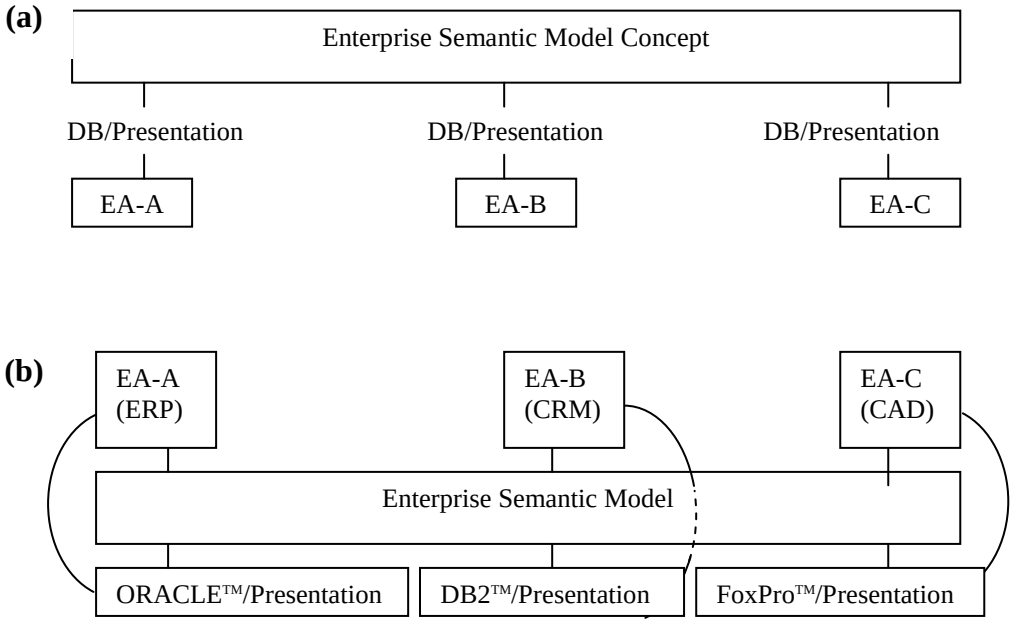
Files/Programs could not communicate with each other automatically prior to the database system concept, which allows them to communicate with each other automatically (or with the help of DBMS). In essence, the DBMS has captured some commonalities in various application data, both in data structure and data manipulation. Different DBMSs have, however, created a new communication barrier. The semantic model concept, then, creates another common platform for different programs/DBMSs to communicate with each other- a means to overcome the new barrier and a sense of integration.

A closer view of the history of the computing world reveals that integration has been an issue that has constantly existed, which reflects freedom in academic research (i.e., divergence) and usability (i.e., convergence). Each time, solutions to integration appear to create a higher layer of platform for representing and managing the commonalities among different participating entities (e.g., files, DBMSs). In the case of evolution of programs/files into programs/databases and into programs/semantic databases, the commonality the semantic model has focussed on is such that all different programs more or less address a part or an aspect of semantics of a discourse under consideration, and hence, the route to integration is along the path of identification of common semantics and provision of computer systems for managing the common semantics.

4.2.3 Analogy leading to the enterprise semantic model (ESM) approach

The present situation of enterprise application integration (EAI) resembles past situations of computer programs integration; enterprise applications (e.g., various

CAD/CAM systems, various ERP systems) are analogous to programs and, further, analogous to programs/databases. This analogy can straightforwardly lead to a semantic model approach to EAI, as shown in Figure 4.2(a). Figure 4.2(b) shows the architecture of operations of these enterprise applications. From this analogy, it becomes clear that an enterprise semantic model (ESM) needs to be first researched.



DB: Database; ERP: Enterprise Resource Planning; CRM: Customer Relationship Management; CAD: Computer Aided Design; EA-A, EA-B, EA-C: Enterprise Application Programs

Figure 4.2 Semantic model approach to EAI

An essential requirement for ESM is that it should capture and represent enterprise semantics. The concept of an enterprise includes the activity (or function) concept, the resource concept, and the organization structure concept. One of the problems with the current practices in EAI, as analyzed in Chapter 1 and Chapter 2, is a missing link between enterprise functions (or activities) and enterprise applications (or information systems). Another problem with EAI is uncertainty due to dynamic evolutions of enterprise applications. To address these two problems, ESM must capture this missing link. This is the important feature of the ESM approach

proposed in this thesis that distinguishes it from existing approaches (see Chapter 2).

4.3 General Methodology for ESM

ESM will be developed as a framework consisting of (i) a set of templates, (ii) guidelines to extend the templates, and (iii) guidelines to instantiate the templates given a particular enterprise. These guidelines are illustrated using examples. An ontology approach will be applied to develop the ESM framework. Ontology in the context of system modeling is a set of fundamental entities and their connectivities within a discourse. Ontology represents things that are the most common and, thus, shareable by different instances of a discourse under study. Semantics of a system will be built upon the ontology of that system. Along this line of thinking, enterprise ontology is a set of things that are common and shareable by enterprises. Ontology modeling refers to a process that leads to a formal representation of captured or defined ontology. In order to capture and represent the missing link in ESM (see the discussions in Section 4.2.3), enterprise ontology modeling will be carried out for both enterprise functions and enterprise applications.

To make ESM as flexible or generic as possible, the data abstraction method called "generalization/specialization" is applied here. This then results in the organization of ESM with the ontology approach in a hierarchical manner (see Figure 4.3). In this lattice, Level 1 contains most generic things about an enterprise. Specializations proceed up to Level 3. At Level 4, a conventional data structure or schema, which uses those defined in the lower levels, can be derived and defined. A semantic model of a particular enterprise is simply an instantiated template (i.e., Level 5). It is noted that such a hierarchical organization has not been seen in the literature. Usually, a two-layer organized structure was employed in the existing studies on ontology [Fox & Gruninger 1998] or meta-modeling for design [Yoshikawa et al. 1994].

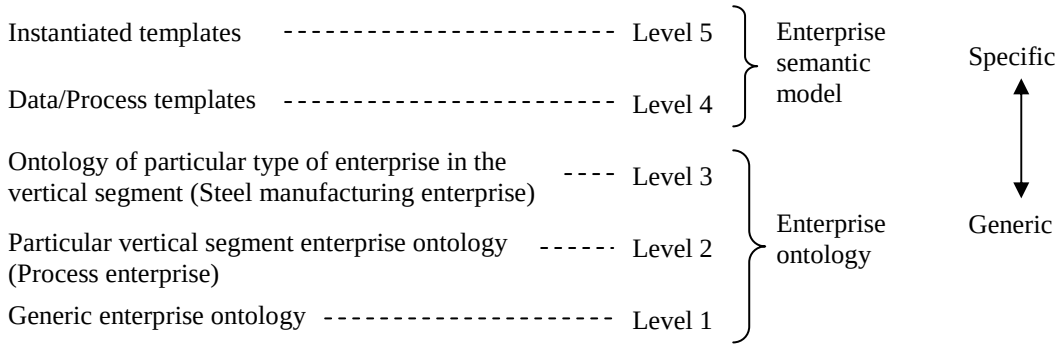


Figure 4.3 Enterprise ontology lattice

As opposed to existing enterprise ontology modeling studies (see Chapter 2), unified modeling language (UML) will be used in this thesis. Concepts and their relationships in enterprise ontology can be expressed using various class and relationship constructs in UML. More dynamic and complex relationships in ontology can be expressed using the object constraint language (OCL) in UML. Things at level 4 and level 5 need more UML constructs (refer to Chapter 3). The remainder of this chapter will present a detailed development of ESM based on the methodology. In particular, Section 4.4 and Section 4.5 will present ontology for enterprise functions and ontology for enterprise applications, respectively; this corresponds to level 1 (Figure 4.3). Section 4.6 presents ontology for a process enterprise (i.e., level 2). Section 4.7 presents further specialized ontology for a steel manufacturing enterprise (i.e., level 3). Section 4.8 shows how templates (conventional database schema) are derived and defined; this corresponds to level 4. On a general note, these developments do not pursue a complete model, but a model with extendibility (or genericity) and practicability (see research objective 2 and the idea described in Figure 1.3).

4.4 Ontology of Generic Enterprise Functions

In this section the three main concepts of an enterprise, namely, activity, resource, and organization structure are modeled. The formalization of activity is crucial in

any attempt to represent an enterprise. **Activities** are events that specify a transformation on the world. **States** enable activities and are also caused by activities. Activities are initiated at some point in **time**, and once initiated, they have **duration** over some interval of time. Further, properties of states hold over the duration of these activities. All activities require that some objects be available at the time that the activity is performed. These objects act as **resources** for the activities. The resource has a **location** and its **quantity** is measured by an appropriate unit. The **organization structure** of an enterprise is divided into many **divisions** and each division has its own **goals**; however, their collective goal is to fulfill the goal of the organization. The organization has many **agents**, and they perform different **roles** to achieve the goals.

4.4.1 Activity ontology

Figure 4.4 represents the activity ontology of an enterprise by using a UML class diagram. The concepts are modeled as classes and the generalization/specialization relationship between the classes are modeled by using an arrow with a hollow triangle end. The class to which the arrow points is the generalized class, and the class from which the arrow originates is the specialized class. 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 will be true of the world once the activity has been completed. There are two types of states, *terminal and non-terminal*. *Use, consume, release and produce* states belong to the terminal state, and *exclusive, not, conjunctive and disjunctive* states belong to non-terminal state. The activity uses the *resource*. Both the activity and state have an attribute called *status*. This attribute is modeled as a class and has two subclasses: state status and activity status. *Possible, committed, enabled, re-enabled, disenabled and completed* are the statuses of the state. *Dormant, terminated, executing, suspended and re-executing* are the statuses of the activity. The status is changed by *action*. *Commit, enable, disenable, re-enable and*

complete are the types of action. The action occurs in a *situation*, and the situation needs *time*.

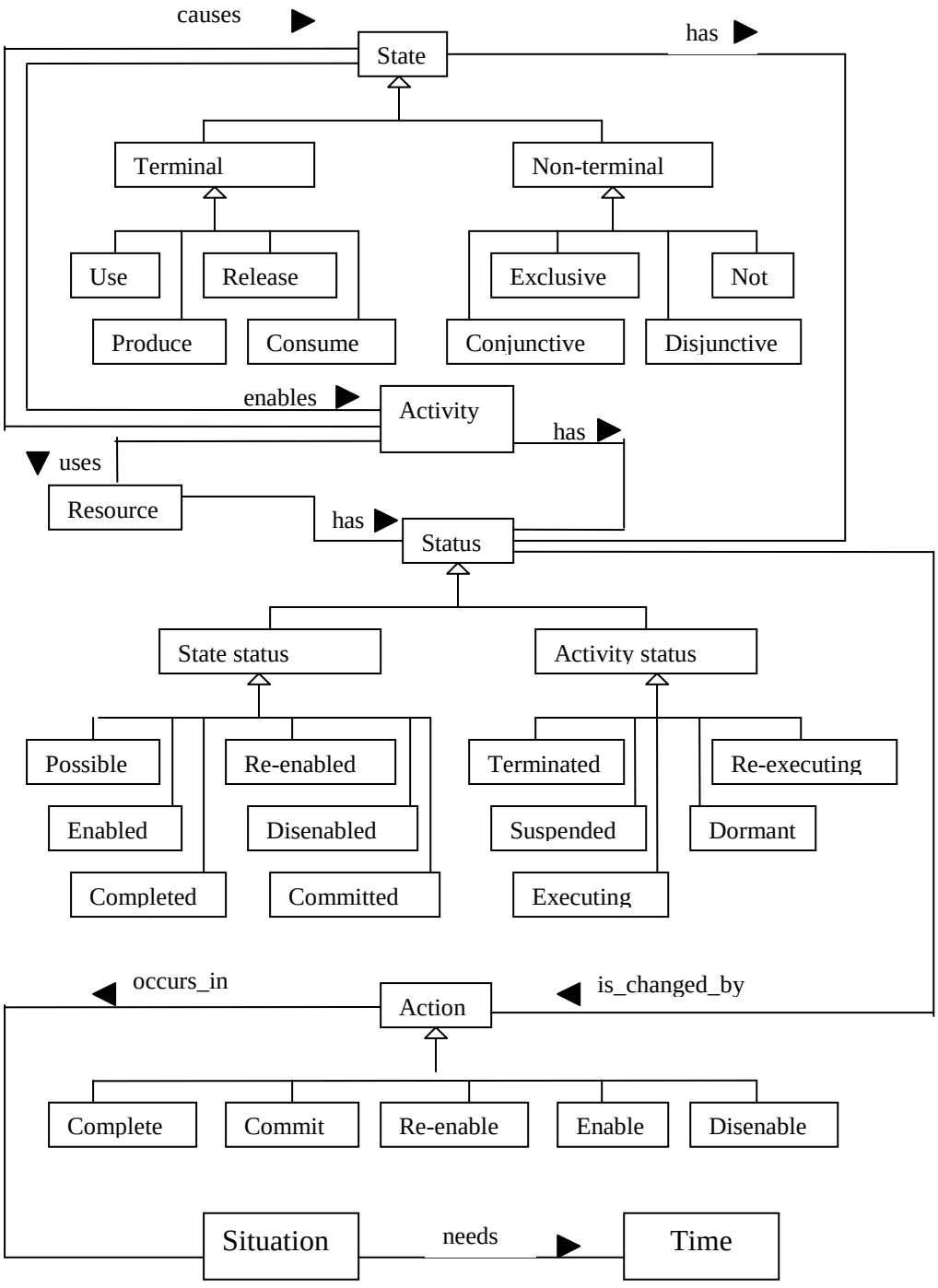


Figure 4.4 Activity ontology

Not all concepts shown in Figure 4.4 can be defined diagrammatically, especially those concepts that are related to logical operations (e.g., sequence, order, list, etc). They can, however, be defined by texts, i.e., object constraint language (OCL) of UML (see Chapter 3 for details). Enable action is defined using OCL below.

> Context State inv:
> Enable
> Pre: Status=Committed
> Post: -
> An action Enable takes place with the pre-condition that the status of the state is committed.

Similarly, the following concepts from Figure 4.4 are defined using OCL and documented in Appendix A:

- Action and time: commit, re-enable, disenable, complete, and duration
- Activity status: terminated, suspended, re-executing, dormant, and executing
- State status: possible, re-enabled, enabled, disenabled, completed, and committed
- Terminal state: use, release, produce, consume
- Non-terminal state: exclusive, not, conjunctive, and disjunctive

4.4.2 Resource ontology

Being a resource is not an innate property of an object but is derived from the role an object plays with respect to an activity. Primitive resource properties are identified and defined (Figure 4.5), from which more complex properties would be defined. An activity uses a resource. A resource has *quantity, location, and role* and is governed by *restriction*. Statuses of a resource are *production, consumption, and use*. A resource can be divided into *continuous and discrete*. A continuous resource is uncountable, whereas discrete resources such as finance, human, and supplier are countable. In Figure 4.5, there can be various discrete resources other than finance,

human or supplier; and hence, the discrete resource class is not completely instantiated. Resources can also be divided into *physically and functionally divisible resources*, and they both have their respective components.

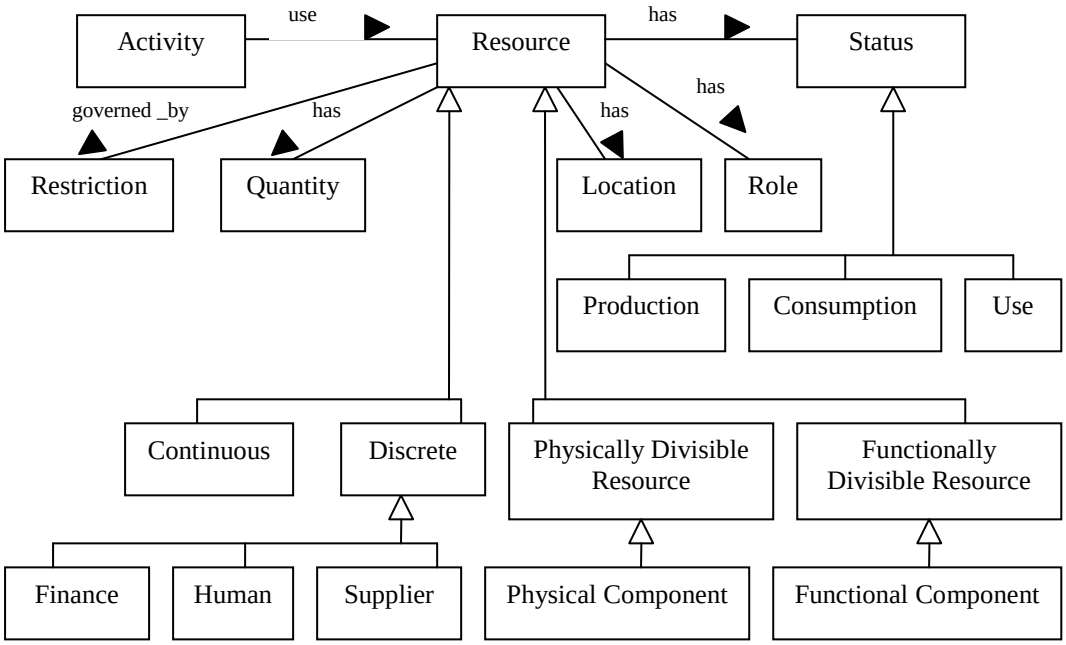


Figure 4.5 Resource ontology

Using OCL, the concept of *consumption* of a resource is defined below.

Context Resource inv:

Consumption

Pre: $q_s.(R.A.u) = true$

Post: $q_s.(R.A.u) > q_e.(R.A.u)$

The consumption specification term entails that the resource amount will be decremented after the completion of the activity, i.e., the quantity q_s of the resource-R at the start of the activity-A, in terms of unit-u, will be greater than the quantity q_e at the end of the activity-A.

It is noted that the definitions of the following concepts can be seen in Appendix B:

- Resource status: use, and production
- Resource type: continuous, discrete, physically divisible, and functionally divisible

Supplier ontology: A supplier provides both products and information. An enterprise receives products and information and pays back to the supplier (see Figure 4.6). The products from the supplier can be of many types, such as manufactured, bought, and non-physical (e.g., service). The supplier provides information about the product it provides. The information the supplier and the enterprise provide each other could be scheduling and resource information within their respective organizations. The information is modeled in UML using a rectangle with angled sides as shown in Figure 4.6.

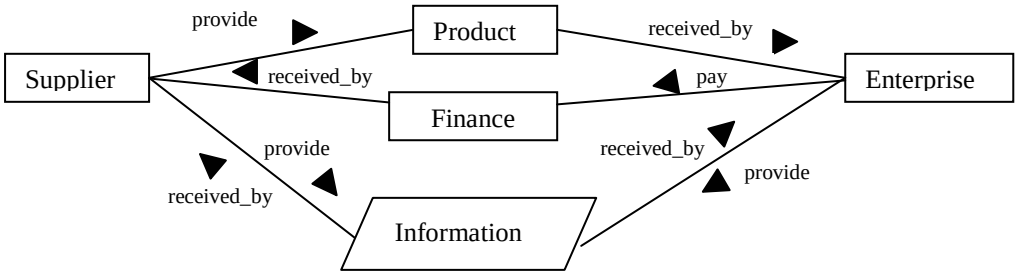


Figure 4.6 Supplier ontology

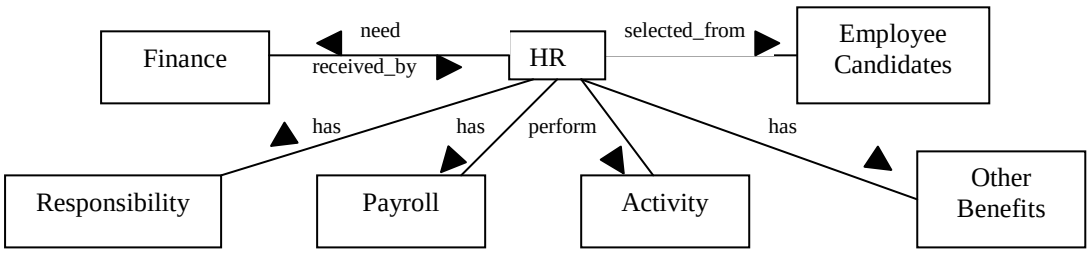


Figure 4.7 Human resource ontology

Human resource ontology: Human resources (HR) are selected from the employee candidates by conducting interviews. The HR performs activity, needs and receives finance, has responsibility, and has payroll and other benefits (see Figure 4.7). The HR acts as agents, which are described in the organization ontology.

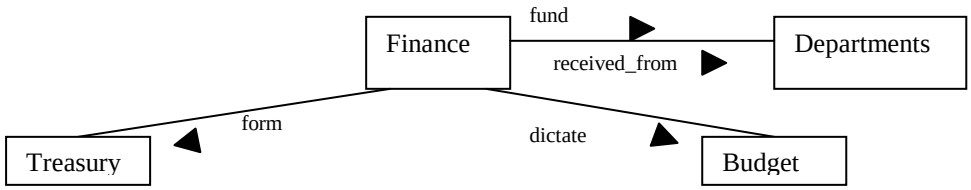


Figure 4.8 Finance ontology

Finance ontology: Finance receives money and in turn funds various departments. Finance forms the treasury and dictates the budget (see Figure 4.8).

4.4.3 Organization ontology

Figure 4.9 represents the organization ontology of an enterprise. An organization is defined as a set of constraints on the activities performed by agents. In particular, an *organization* consists of a set of *divisions* and has some *goals.* Each division and goal has a set of *sub-divisions* and *sub-goals,* respectively. There are *organization-agents* belonging to each division or sub-division. An agent can also be a member of a *team* set up in response to a special task. Each agent has a set of *communication-link* defining the protocol, based on which that agent communicates with other agents in the organization. Each agent has a set of *roles,* and each role is defined with a set of *goals* the role is created to fulfill. Each role requires certain *skill* and is allocated with proper *authority* at the level that the role can achieve its goals. Agents perform *activities* in the organization, each of which may consume *resources,* and there is a set of *constraints* that restrict the activities.

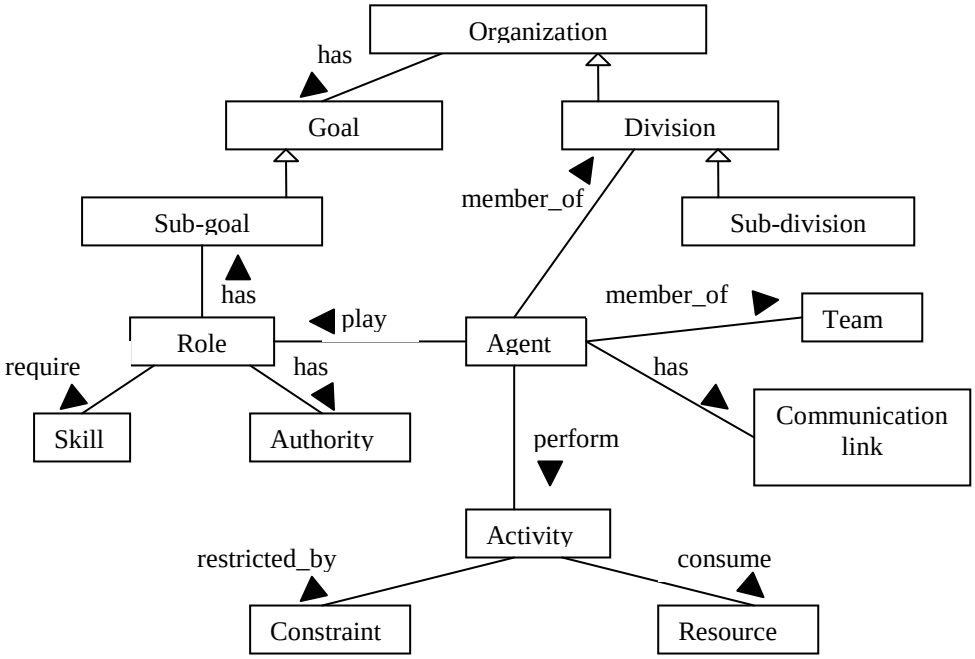


Figure 4.9 Organization structure ontology

Using OCL, the concept of a *role* having an *authority* is defined below.

Context Role inv:

r.at implies role *r* has authority *at*

It is noted that the definitions of the following concepts can be seen in Appendix C:

- Organization structure: role, goal, and agent

4.5 Ontology of Generic Enterprise Information Systems

It should be noted that for the purpose of EAI, this thesis research treats an enterprise application (i.e., an information system) as a black box with some interfaces that are essential for communication with other applications. These interfaces facilitate communications, particularly in terms of (i) functionality, (ii)

information transfer, and (iii) physical compatibility (e.g., operating systems). From the EAI viewpoint, the information of these interfaces is apparently more important than the structure of the software.

Ontology of the storage and retrieval of data:

Figure 4.10 represents the storage and retrieval of data from a database. The client inputs the data to be stored or queried using input devices like a keyboard in an user interface. These data or queries are received by the application server where the application logic is present. The application server formats the data and sends them to the database server to be stored in the database. The database server then stores these data in an appropriate place in the database. In case of queries, the application server identifies the appropriate data to be retrieved for the queries and sends instructions to the database server about the data to be retrieved. The database server then retrieves the data to be retrieved as instructed by the application server from the database and sends them to the application server, which then sends them to the user interface for "showing" to other applications. The database server saves all the information about the data retrieved and the location of the data stored in the database. The physical database is a destination for the data.

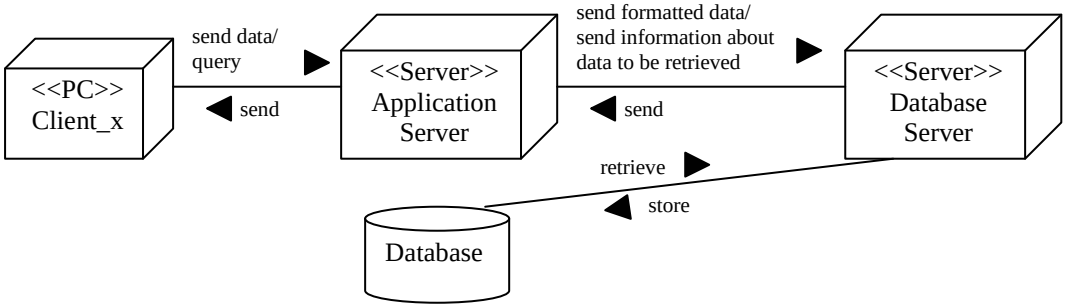


Figure 4.10 Ontology of the storage and retrieval of data

Ontology of the functional view information systems:

An enterprise has various information systems. These information systems can be categorized into three main classes, corporate, execution, and management systems.

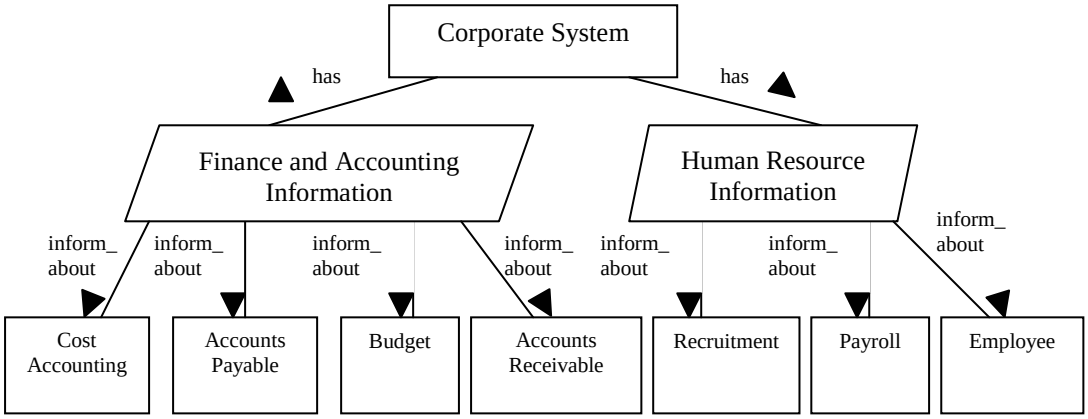


Figure 4.11 Corporate information system ontology

Corporate information system ontology: Corporate information systems (Figure 4.11) have information regarding finance and accounting and human resources (HR). Finance and accounting information informs about cost accounting, budget, accounts payable, and accounts receivable. Human resource information informs about recruitment, payroll, and employee information.

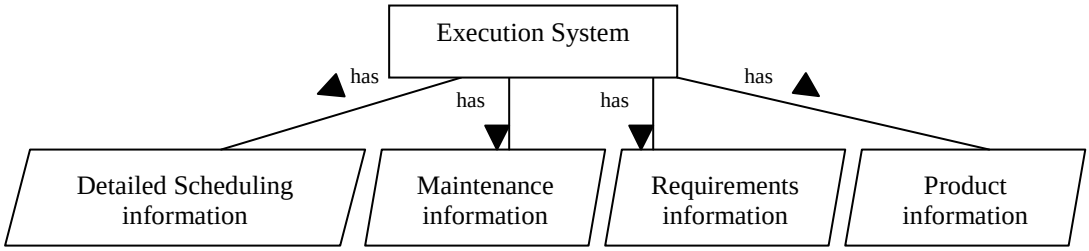


Figure 4.12 Execution information system ontology

Execution information system ontology: Execution systems (Figure 4.12) constitute systems that execute the core function of an enterprise. They have information regarding scheduling, maintenance, requirements, and product.

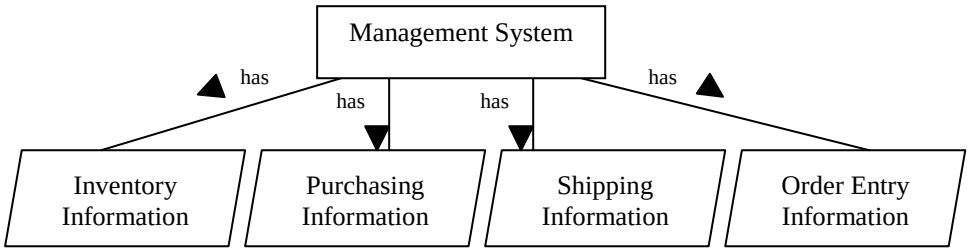


Figure 4.13 Management information system ontology

Management information system ontology: Management systems (Figure 4.13) have information regarding inventory, purchasing, shipping, and order entry.

Ontology of information system- cost:

The information system within an enterprise has some costs associated with it (Figure 4.14).

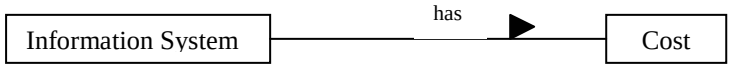


Figure 4.14 Ontology of information system- cost

Ontology of information system- software interface:

The information system within an enterprise has some software interface (Figure 4.15). The software interface has some costs associated with it.

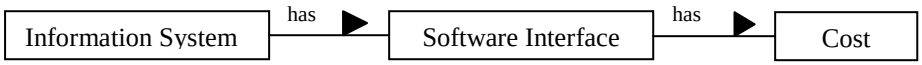


Figure 4.15 Ontology of information system-software interface

Ontology of information system- hardware interface:

The information system within an enterprise has some hardware interface (Figure 4.16). The hardware interface has some costs associated with it.

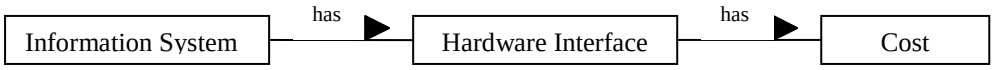


Figure 4.16 Ontology of information system- hardware interface

Ontology of information system- operating system (OS):

The information system within an enterprise has an operating system (Figure 4.17). The OS has some costs associated with it.

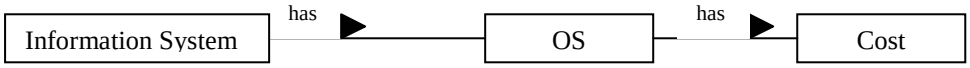


Figure 4.17 Ontology of information system- OS

Ontology of information system- database management system (DBMS):

The information system within an enterprise stores its data in the database using a database management system (Figure 4.18). The DBMS has some costs associated with it.

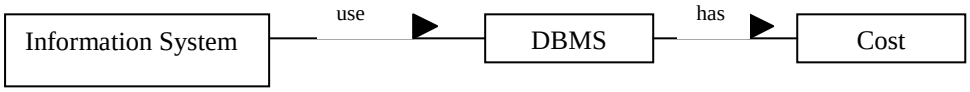


Figure 4.18 Ontology of information System- DBMS

Ontology of information system- manufacturer's information:

The information system within an enterprise has a manufacturer (or producer) (Figure 4.19). There are various attributes of information regarding the manufacturer, which are used to evaluate the manufacturer and compare the manufacturer with other manufacturers on the market. Supplier selection decisions are made using these information.

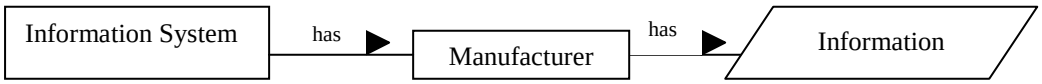


Figure 4.19 Ontology of information system- manufacturer's information

4.6 Specialization of Generic Enterprise Ontology (GEO) into Generic Process Enterprise

In this section the generic enterprise ontology (GEO) described in Section 4.4 and 4.5 will be extended for a particular type of enterprise, a process enterprise. The process enterprise provides a unique challenge as it is a continuous type of enterprise. Production is not altogether stopped; production can, however, be slowed down for grade changes or unusual operating conditions. This is because stopping the production for overhaul/restart can be very expensive and/or time consuming. The purpose of this section is also to show how the hierarchical organization of ESM works (see Figure 4.3).

4.6.1 Ontology of generic process enterprise functions

In this section, the generic enterprise functions presented in Section 4.4 will be specialized into generic process enterprise functions.

Business process ontology:

Engineering ontology: The engineering or design process is elaborated in Figure 4.20. The engineering process is done in accordance with its quantitative and qualitative goals and is driven by the deadlines and controlled by the design manager. The engineering process takes as inputs the necessary data (e.g., specifications) and produces the analysis results. It uses the design personnel and the information technology necessary to perform its function.

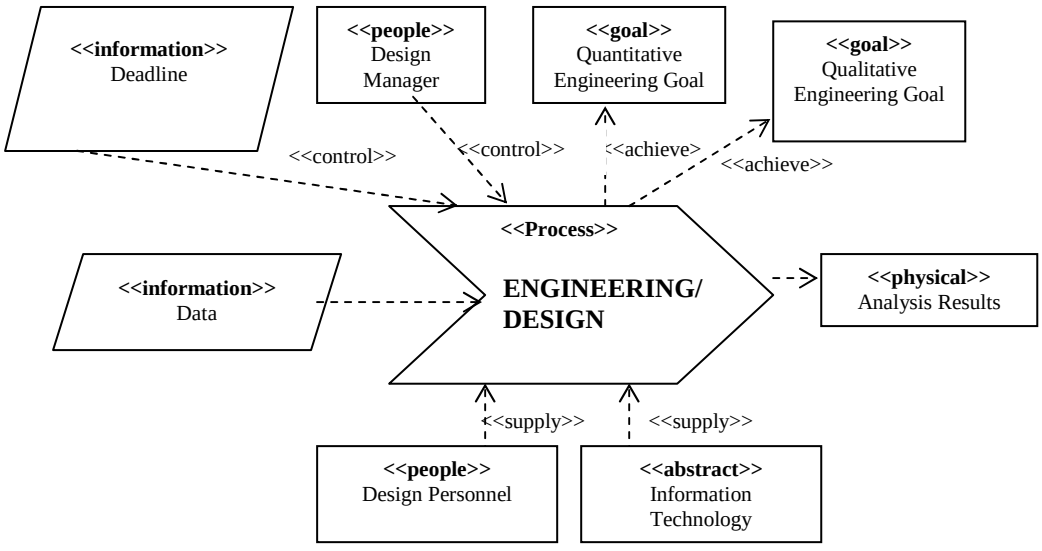


Figure 4.20 Engineering ontology

Manufacturing ontology: The manufacturing process is elaborated in Figure 4.21. The manufacturing process is done in accordance with its quantitative and qualitative goals and is driven by deadlines and controlled by the production manager. The manufacturing process takes as inputs raw materials and produces

finished goods. It uses the manufacturing personnel and the manufacturing equipment necessary to perform its function.

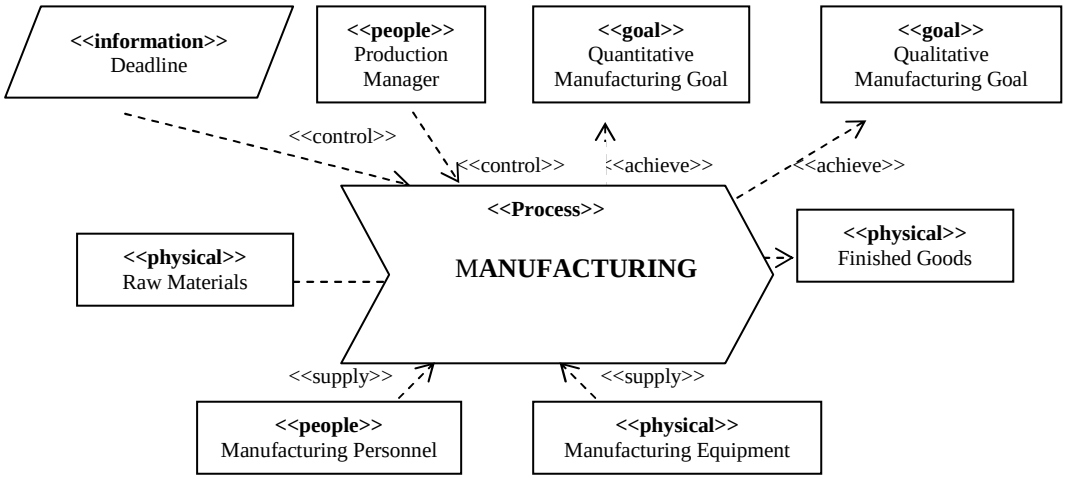


Figure 4.21 Manufacturing ontology

Resource ontology:

Employee ontology: Employees (Figure 4.22) perform some activities and have a department. Employees have salary and benefits, working schedules, and records.

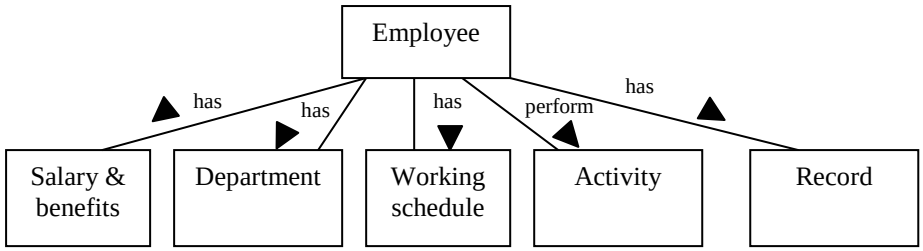


Figure 4.22 Employee ontology

Equipment ontology: Equipment (Figure 4.23) is used to manufacture products and is maintained by employees. The equipment belongs to a department, has a value, and has a working schedule and a working condition.

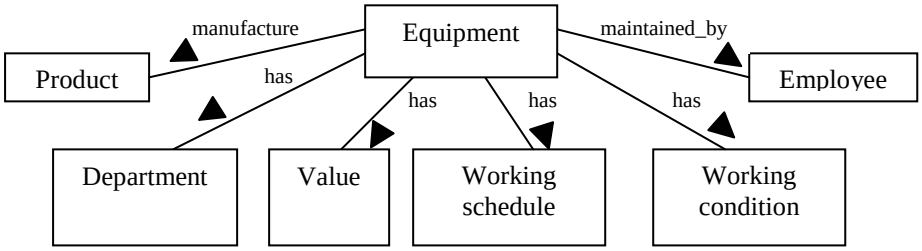


Figure 4.23 Equipment ontology

Finance ontology: Finance (Figure 4.24) is received from banks and buyers. Finance is used to pay for various expenses of the department and salaries. Finance has accounts and interest.

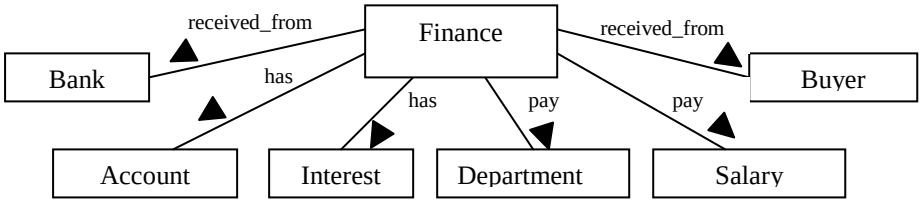


Figure 4.24 Finance ontology

Organization structure ontology:

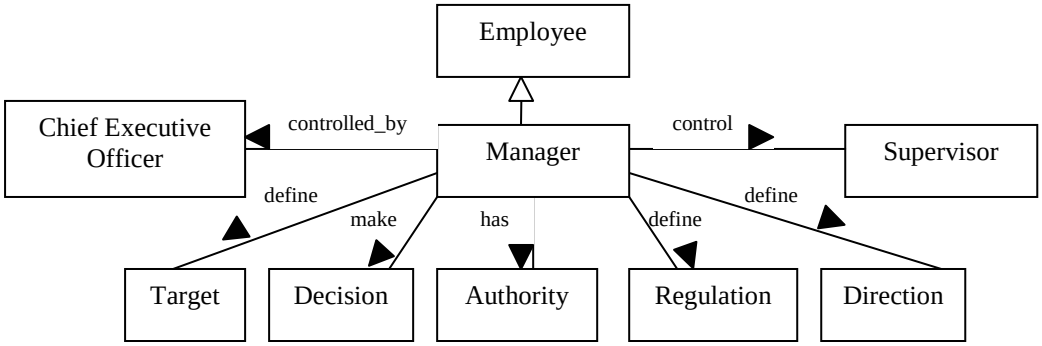


Figure 4.25 Manager ontology

Manager ontology: A manager (Figure 4.25) is an employee, controlled by a chief executive officer. The manager controls the supervisor, defines targets, makes decisions, has authority, and defines regulations and direction.

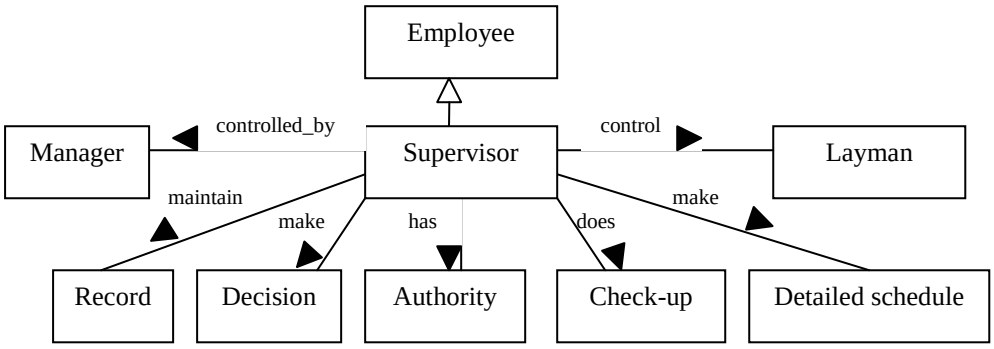


Figure 4.26 Supervisor ontology

Supervisor ontology: A supervisor (Figure 4.26) is an employee, controlled by the manager. The supervisor controls the laymen, maintains records, makes decisions, has authority, does check-ups, and makes detailed schedules.

Supplier's information ontology:

A supplier of a generic process enterprise (Figure 4.27) has various types of information, which are used in collaborative design, collaborative manufacturing, supplier selection, supplier rating, and decision making.

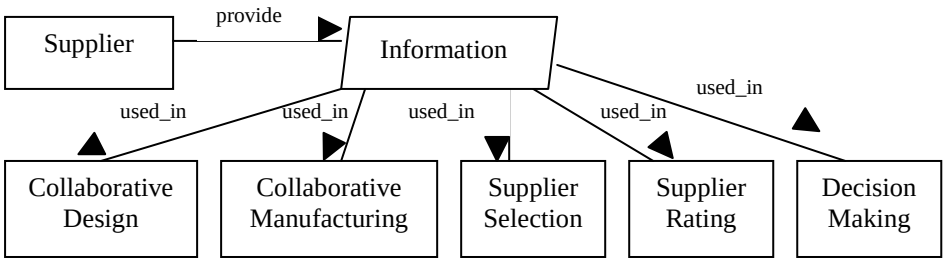


Figure 4.27 Supplier's information ontology

4.6.2 Ontology of generic process enterprise information systems

In this section, the generic enterprise information systems modeled in Section 4.5 will be specialized into generic process enterprise information systems.

Ontology of the functional view of information systems:

In a generic process enterprise, there are five main categories of information systems, corporate, engineering, management, execution, and process control systems.

Engineering information system ontology: Engineering information systems (Figure 4.28) have information regarding quality, schedule, design, and requirements.

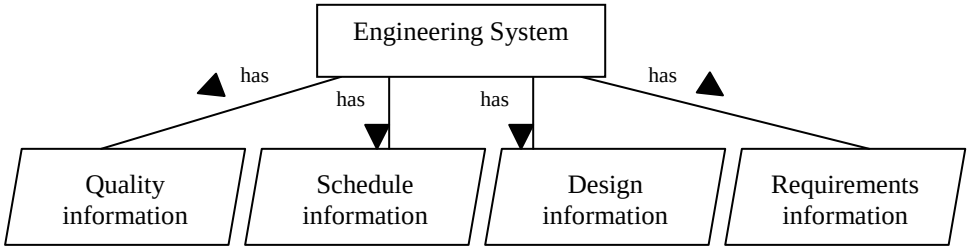


Figure 4.28 Engineering information systems ontology

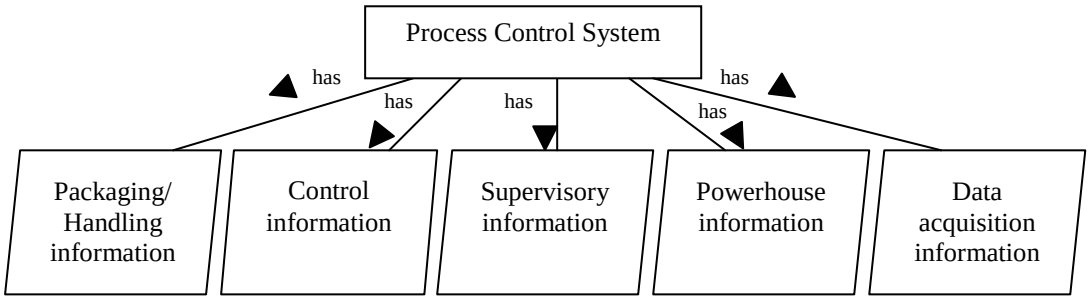


Figure 4.29 Process control information systems ontology

Process control information system ontology: Process control systems (Figure 4.29) have information regarding packaging/handling, control, supervision, powerhouse, and data acquisition.

Ontology of information system- cost:

Information systems of a generic process enterprise have some costs (Figure 4.30), which are affected by the supplier, compatibility, adaptability, usability, and hardware.

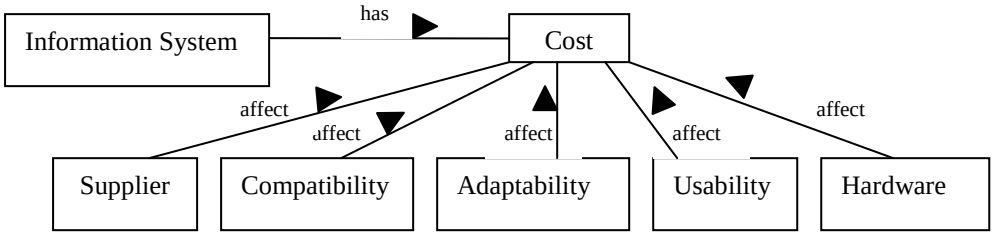


Figure 4.30 Ontology of information system- cost

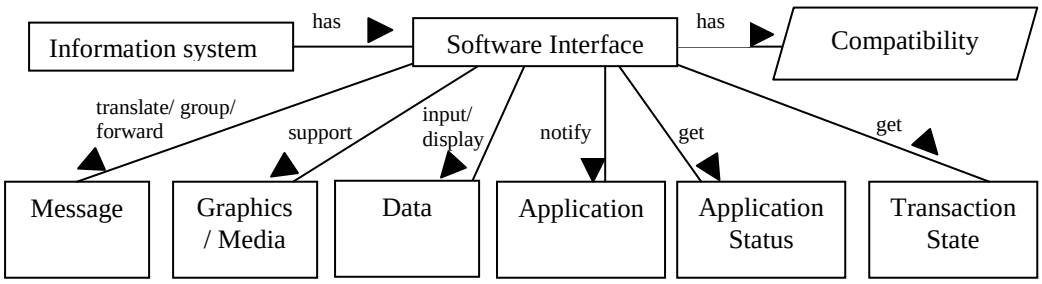


Figure 4.31 Ontology of information system- software interface

Ontology of information system- software interface:

Generic process enterprise information systems have various types of software interfaces (Figure 4.31). The interfaces have compatibility issues. The interfaces

translate, group, and forward messages, notify applications, get application statuses, get transaction states, support graphics/media and input/display data.

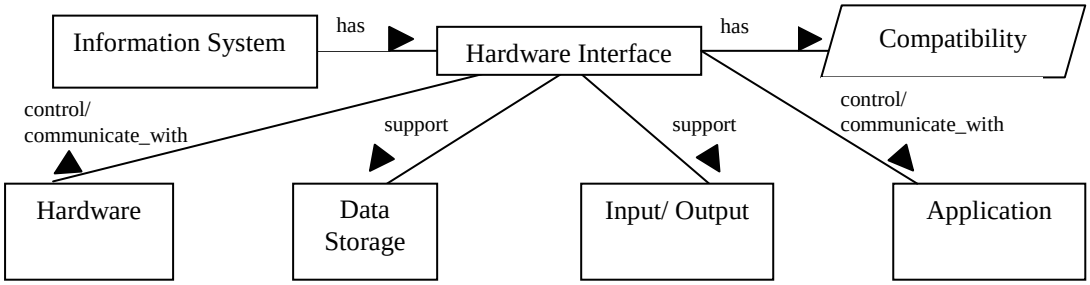


Figure 4.32 Ontology of information system- hardware interface

Ontology of information system- hardware interface:

Generic process enterprise information systems have various types of hardware interfaces (Figure 4.32). The interfaces have compatibility issues. The interfaces control and communicate with applications and hardware, support data storage, and support input/output.

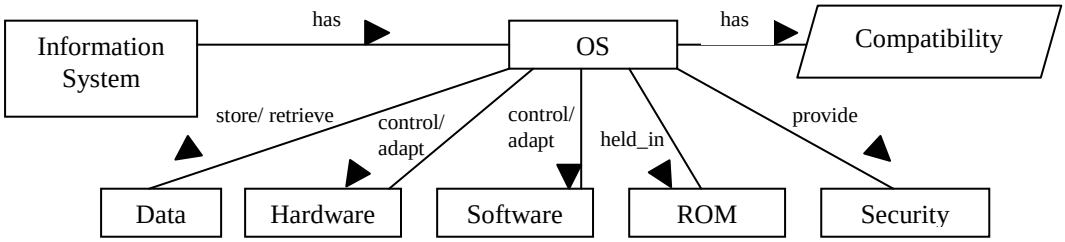


Figure 4.33 Ontology of information system- OS

Ontology of information system- operating system (OS):

Generic process enterprise information systems have various types of OS (Figure 4.33). The OS has compatibility issues, stores/retrieves data, controls/adapts hardware/software, held in read-only memory (ROM), and provides security.

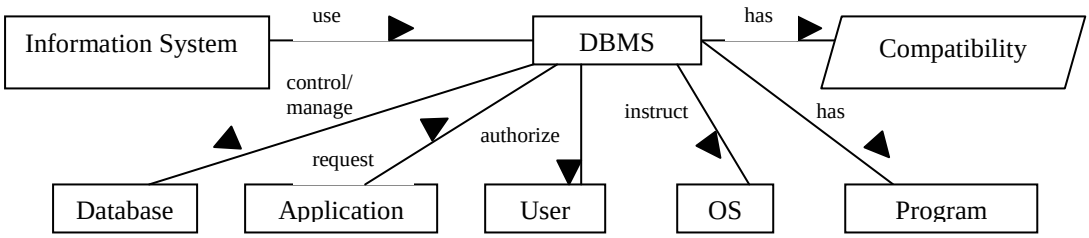


Figure 4.34 Ontology of information system- DBMS

Ontology of information system- database management system (DBMS):

Generic process enterprise information systems use various types of DBMS (Figure 4.34). The DBMS has compatibility, controls/manages database, receives requests from applications, authorizes users, instructs OS, and has programs.

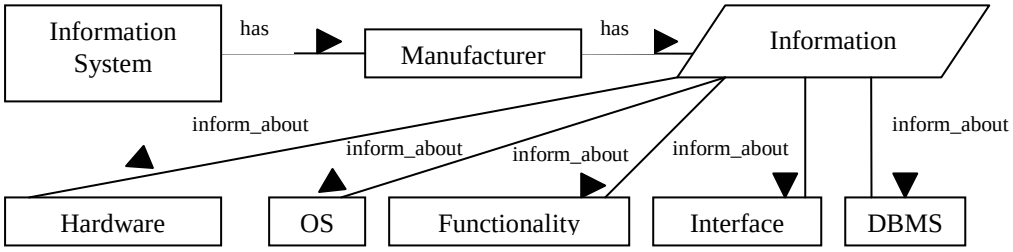


Figure 4.35 Ontology of information system- manufacturer's information

Ontology of information system- manufacturer's information:

An information system manufacturer of a generic process enterprise has information (Figure 4.35) that informs about various aspects of its products, such as hardware, functionality, operating system, interface, and DBMS information.

4.6.3 Link between the generic enterprise ontology (GEO) and generic process enterprise ontology (GPEO)

It is noted that GEO was established in Section 4.4 and Section 4.5, respectively. In Section 4.3, a methodology was proposed and was illustrated in Figure 4.3. This

methodology is an application of a data abstraction called generalization/ specialization, which is one of the bases for the popular methodology in software engineering, called the object-orientation paradigm. The biggest advantage of organizing information (knowledge and data) in this way is that information at a more generalized level can be reused by objects at specialized levels. This section aims to provide an explanation on the generalization/specialization linkage between GEO and GPEO.

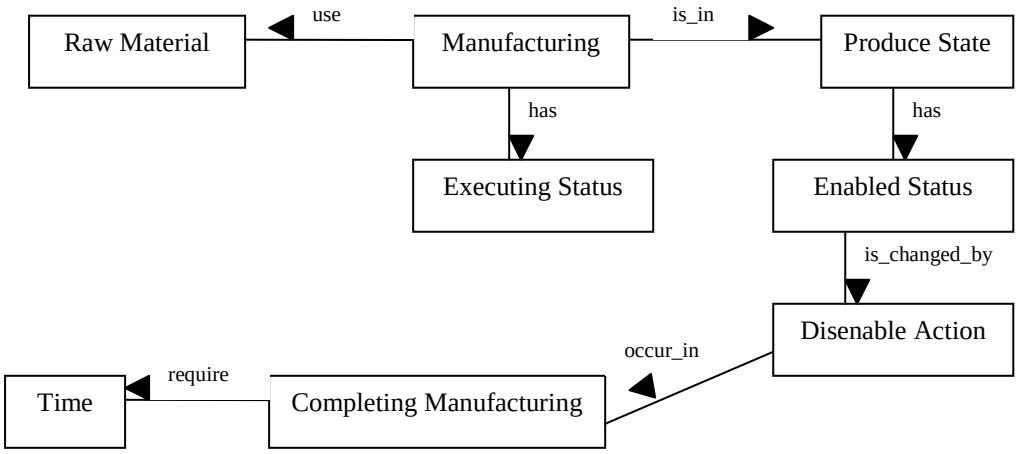


Figure 4.36 Link between GEO for activity and GPEO for activity

Link between GEO for activity and GPEO for activity (manufacturing):

Manufacturing is an *activity*. In Figure 4.36, manufacturing uses raw materials, the manufacturing activity is considered to be in *produce state*, and the status of the produce state is *enabled*. Then the status of the manufacturing activity becomes *executing*. The enabled status of the produce state is changed by *disenable action*; and this action occurs in the situation, *completing manufacturing*. Completing manufacturing requires time. It can be noted here that knowledge at the GPEO level is derived from knowledge at the GEO level by following the link, i.e., GPEO "is-a" GEO; in particular, manufacturing is an activity.

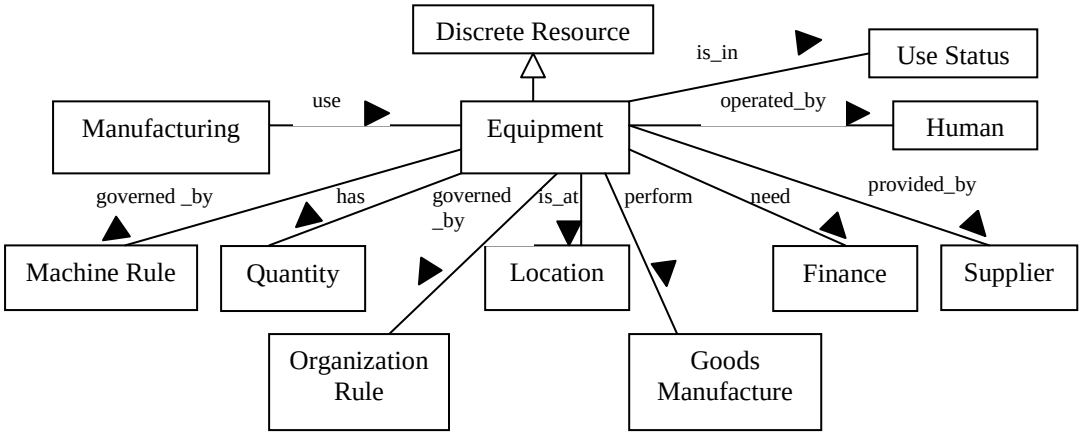


Figure 4.37 Link between GEO for resource and GPEO for resource

Link between GEO for resource and GPEO for resource (equipment):

Equipment is a *discrete resource*. In Figure 4.37, when the manufacturing activity uses the equipment, the equipment is in *use status*. The equipment is operated by humans, provided by the supplier, needs finance, is at a location, is governed by machine rules and organization rules, has quantity, and performs the role of goods manufacture.

Link between GEO for organization structure and GPEO for organization structure (manager):

A m*anager* is a part of the *organization structure*. In Figure 4.38, the manager is considered to belong in the manufacturing department, which is one of the divisions of the organization. The manager can be a member of the design team and communicates through orders. The manager plays the role of a production manager in this case, whose goal is to manufacture goods, which is a sub-goal of the organization goal of making profit. The production manager role requires engineering skills and authorizes production. The Manager performs manufacturing activity, which is restricted by deadlines and consumes raw materials.

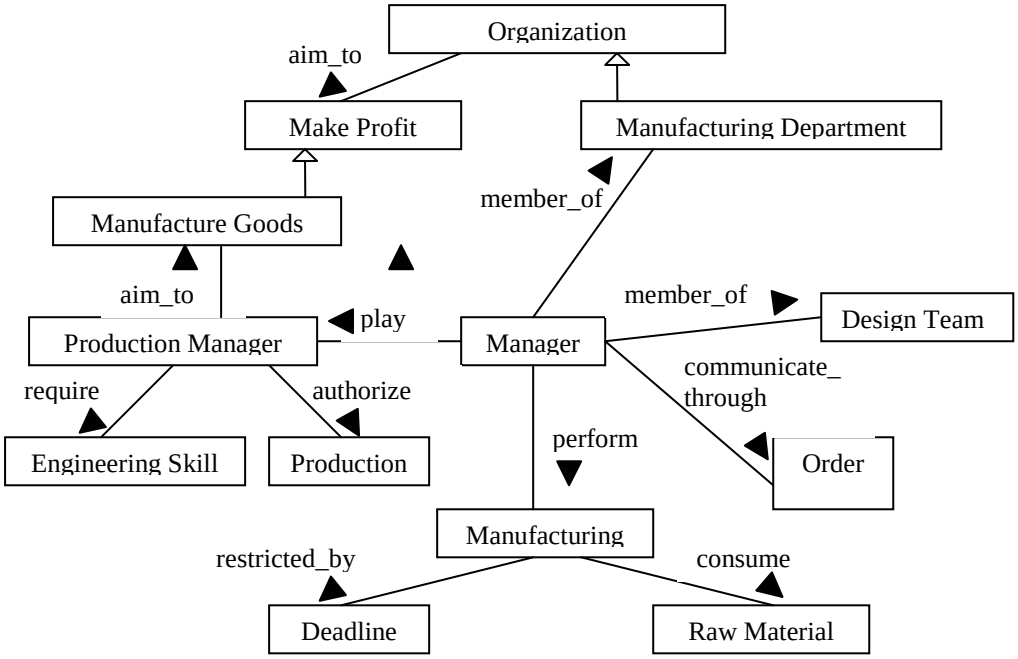


Figure 4.38 Link between GEO for organization structure and GPEO for organization structure

Link between GEO for supplier's information and GPEO for supplier's information:

In Section 4.4 (see also Figure 4.6), it was noted that the supplier provides information to an enterprise. In Section 4.6.1 (see also Figure 4.27), this ontology is specialized to show how the supplier's information is used in the generic process enterprise.

Link between GEO for information system and GPEO for information system:

The information system in a process enterprise is a specialization of the information system in a generic enterprise. In the GEO for information systems section (Section 4.5), it was noted that an information system has cost, a software interface, a hardware interface, an OS, a DBMS, and the manufacturer's information. In the

GPEO for information systems section (Section 4.6.2), the above-mentioned concepts are specialized into the process enterprise. In Section 4.5, the ontology of the functional view of generic enterprise information systems, such as corporate, execution, and management systems, was defined. In Section 4.6.2, the ontology of the functional view of information systems specific to a process enterprise, such as engineering and process control systems, was defined. It is noted that the generic information systems defined in Section 4.5 were re-used in Section 4.6.2.

4.7 Specialization of GEO and GPEO into Generic Steel Enterprise

In this section, the GEO and GPEO described in the earlier sections will be specialized for a particular type of process enterprise, the steel enterprise. This section also shows how the hierarchical organization of ESM works (see Figure 4.3).

4.7.1 Ontology of generic steel enterprise functions

In this section, the GEO and GPEO, defined in Section 4.4 and Section 4.6.1, respectively, will be customized for generic steel enterprise functions.

Business process ontology:

Manufacturing ontology: The generic steel manufacturing process is elaborated in Figure 4.39. The manufacturing process is done in accordance with its quantitative and qualitative goals and is controlled by deadlines and the production manager. The manufacturing process takes as inputs the raw materials and alloys, and outputs the finished steel goods. It uses manufacturing personnel and manufacturing equipment such as buckets, agents, ladles, furnaces, rolling mills, and cooling beds, as resources to perform its function.

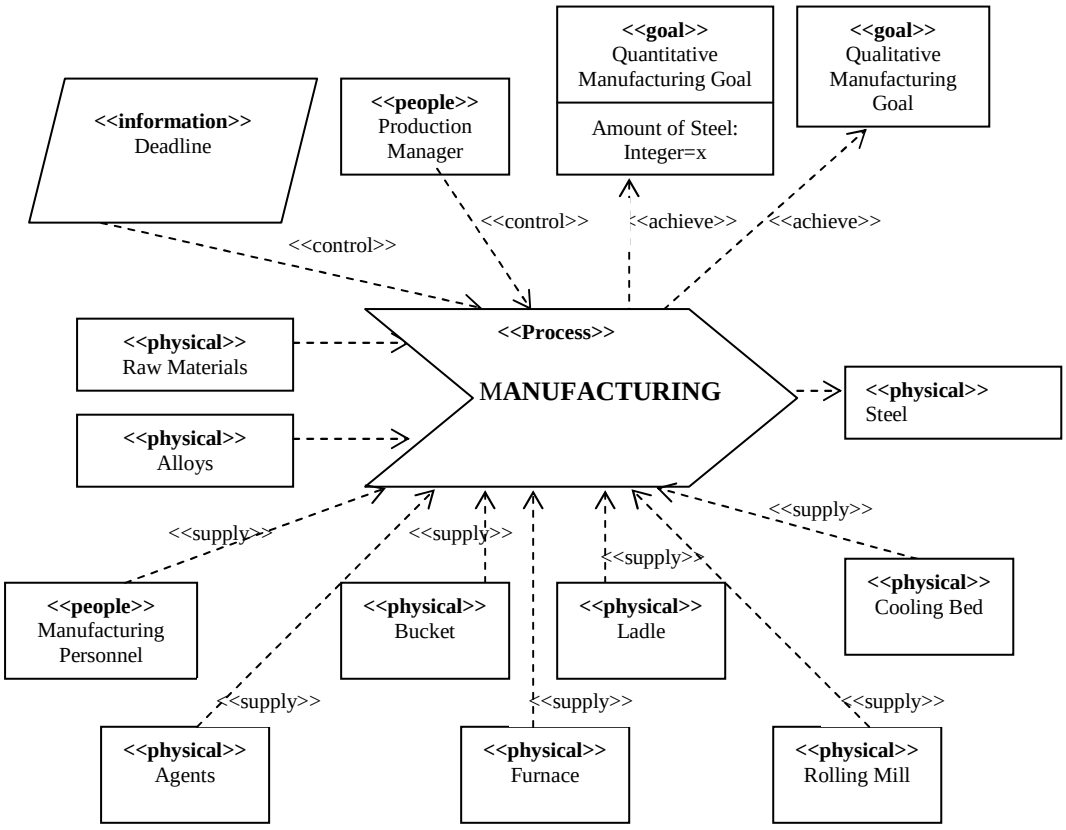


Figure 4.39 Manufacturing ontology

Resource ontology:

Furnace ontology: The furnace (Figure 4.40) is a resource used by a steel enterprise. The furnace melts charge and produces melted steel and slag. Agents and power are used by the furnace and it is controlled by the control system.

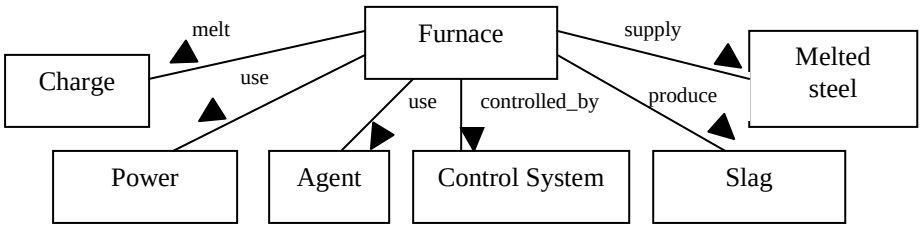


Figure 4.40 Furnace ontology

Organization structure ontology:

Production manager ontology: The production manager (Figure 4.41) is controlled by the plant manager. The production manager co-ordinates resources and activities, plans schedules, monitors standards, analyzes limitations and constraints, reviews recommendations, and controls production supervisors.

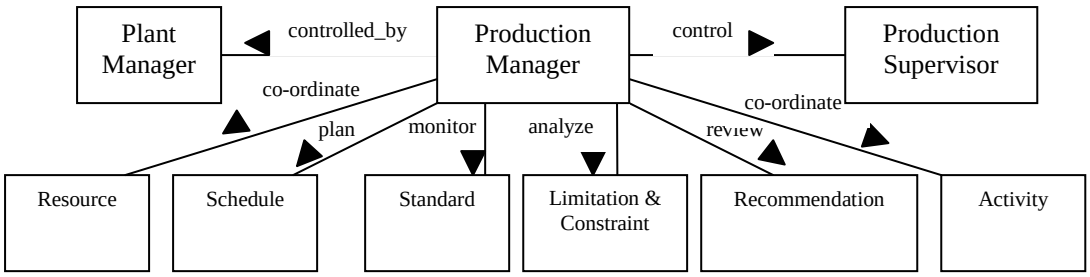


Figure 4.41 Production manager ontology

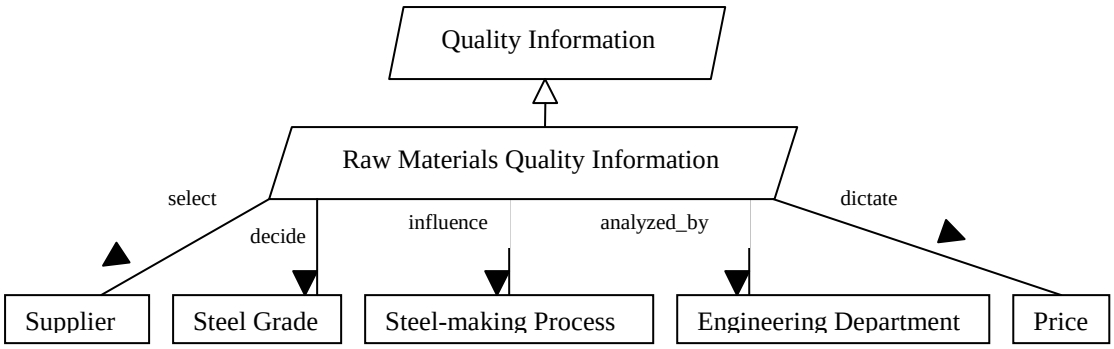


Figure 4.42 Supplier's information ontology

Supplier's information ontology:

The supplier of a steel enterprise has much information to be examined. For example, the raw materials quality information of a supplier (Figure 4.42), is

analyzed by the engineering department, which selects the supplier, decides the final steel grade, influences the steel-making process, and dictates the price.

4.7.2 Ontology of generic steel enterprise information systems

In this section, the GEO and GPEO defined in Section 4.5 and Section 4.6.2 respectively, will be customized for generic steel enterprise information systems.

Ontology of the functional view of information systems:

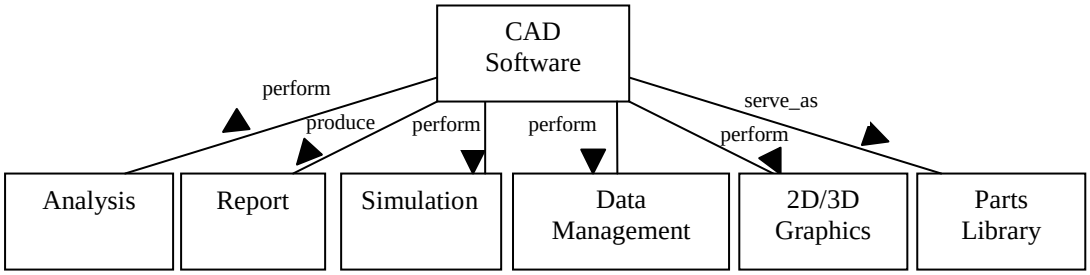


Figure 4.43 Ontology of CAD software

Ontology of computer aided design (CAD) software: A steel enterprise has many information systems for different functions. For example, the CAD software (Figure 4.43) performs analysis, simulation, data management, and 2D/3D graphics; produces reports; and serves as a parts library.

Ontology of information system- cost:

A steel enterprise has many information system related costs (Figure 4.44), such as maintenance cost, associated with it. Maintenance cost is affected by qualified staff; standardized programming language, operating system, and documentation; and system structure.

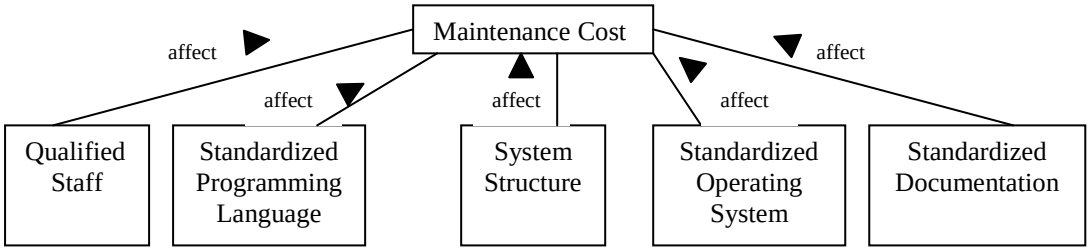


Figure 4.44 Ontology of information system- cost

Ontology of information system- software interface:

A steel enterprise has different information systems with different interfaces (Figure 4.45). For example, the CAD system interface has a toolbar, settings menu, and pop-up menu; and can create/edit/move objects, display online help, control object precision, and change object properties.

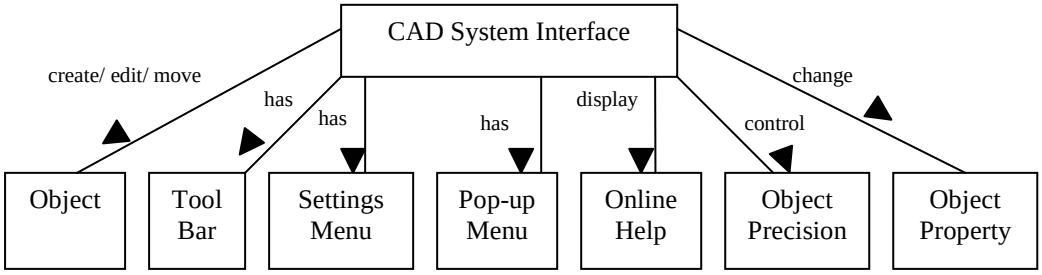


Figure 4.45 Ontology of information system- software interface

Ontology of information system- hardware interface:

A generic steel enterprise has different hardware interfaces. For example, the process control system interface (Figure 4.46) communicates with process control, operator control, and data analysis application; handles signals; and adapts process control equipment.

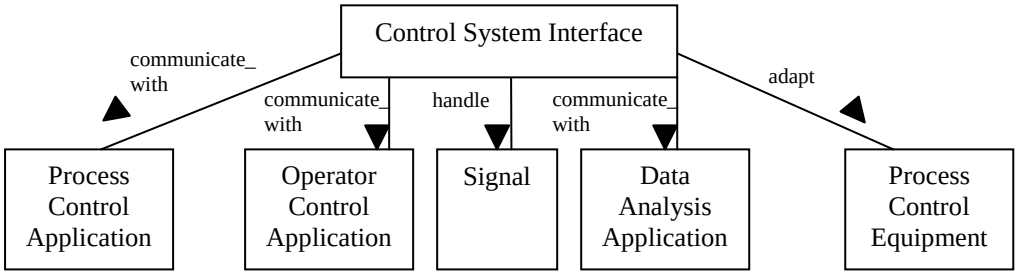


Figure 4.46 Ontology of information system- hardware interface

Ontology of information system- manufacturer's information:

There is much information to be considered regarding the manufacturer of a generic steel enterprise information system (Figure 4.47), including DBMS information. DBMS information informs about the compatibility, security, data transfer, query builder, and space management of the DBMS.

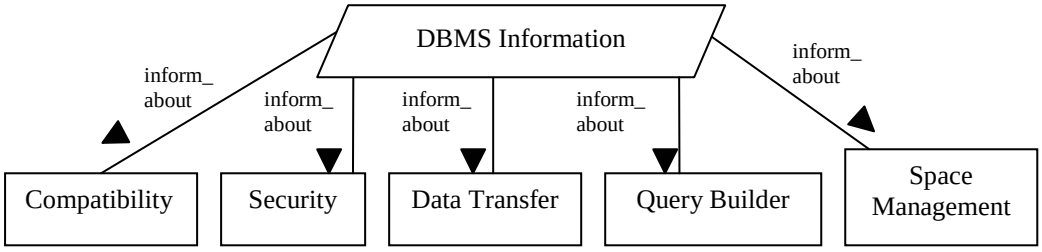


Figure 4.47 Ontology of information system- manufacturer's information

4.7.3 Link between the generic enterprise ontology (GEO), generic process enterprise ontology (GPEO) and generic steel enterprise ontology (GSEO)

It is noted that GEO was established in Section 4.4 and 4.5, and GPEO established in Section 4.6. This section aims to provide an explanation about the generalization/ specialization linkage between GSEO and, GEO and GPEO.

Link between GEO and GPEO for functions, and GSEO for functions:

Manufacturing for a steel enterprise (Figure 4.39) is a specialization of the manufacturing for process enterprise (4.21), and both types of manufacturing are activities (Figure 4.4). The furnace (Figure 4.40) is equipment (Figure 4.23) and eventually a resource (Figure 4.5). Hence, all the concepts that are applicable to resource and equipment are also applicable to the furnace. The production manager (Figure 4.41) is a manager (Figure 4.25) and acts as an agent in the organization structure (Figure 4.9). The production manager is eventually a human resource (Figure 4.7). Hence, all the concepts that are applicable to a human resource, agent, and manager are also applicable to the production manager. The supplier information specific to steel enterprise, raw materials quality information (Figure 4.42), is a specialization of supplier information used in the generic process enterprise (Figure 4.27). Both types of supplier information are developed from the supplier information ontology (Figure 4.6).

Link between GEO and GPEO for information systems, and GSEO for information systems:

In Section 4.6.2, cost, software interface, hardware interface, and manufacturer's information for information systems specific to process enterprise was derived from Section 4.5, where the same concepts generic to all enterprises were defined. In Section 4.7.2, the above mentioned concepts are specialized for information systems specific to a steel enterprise: maintenance cost from generic cost, CAD system interface from software interface, control system interface from hardware interface, and DBMS information from generic information system manufacturer's information. It is noted that while modeling the ontology for the functional view of information systems, the CAD system modeled as information systems specific to steel enterprise is a specialization of the engineering system modeled as information systems specific to a process enterprise. The CAD system re-uses the concepts

developed in the engineering system ontology, and they both re-use the generic information systems defined in the GEO section (Section 4.5).

4.8 Enterprise Semantic Model (ESM) Template

The ESM developed in this section acts as a template, which can be customised to develop the semantic model for a particular enterprise. ESM templates for both enterprise functions and enterprise information systems are developed in this section using the ontologies developed in the previous sections.

4.8.1 ESM template for enterprise functions

In this section, the ESM template is developed for enterprise functional aspects such as business processes, enterprise resources, organization structure, and supplier information.

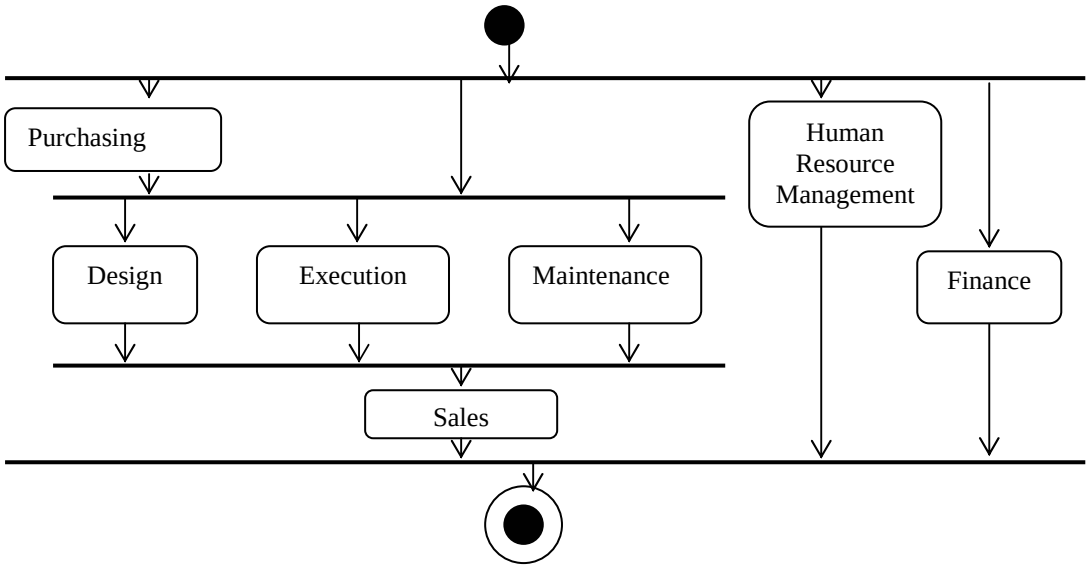


Figure 4.48 ESM template for business processes of enterprise

ESM template for business processes of enterprise:

The ESM template for business processes of enterprise is depicted in Figure 4.48 using an 'activity diagram' of UML. The human resource ontology is defined in Figure 4.7 and the finance ontology in Figure 4.8. Similarly, the ontologies for design, execution, maintenance, purchasing, and sales can be defined. These ontologies are used in the ESM template for business processes of an enterprise, as shown in Figure 4.48. Design, execution, and maintenance occur simultaneously, but only after the completion of the purchasing activity. Sales takes place after all the above-mentioned activities are completed. Human resource management and finance take place simultaneously with all the activities.

ESM template for enterprise resources:

The ESM template for enterprise resources is depicted in Figure 4.49. Employee uses knowledge, materials, equipment, and technology. Equipment uses or makes materials and uses technology. Finance is needed by most of the resources such as employees, materials, equipment, and technology.

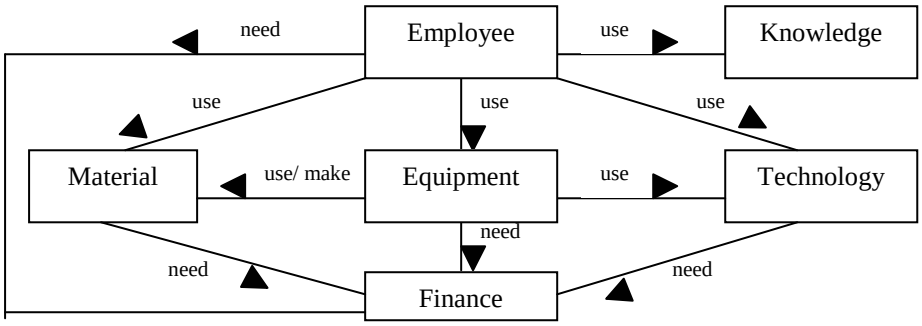


Figure 4.49 ESM template for enterprise resources

ESM template for enterprise organization structure:

The ESM template for enterprise organization structure is depicted in Figure 4.50. The corporate executive officer controls the plant manager, who in turn controls the

execution, sales, purchasing, and finance managers. These department managers control their respective workers.

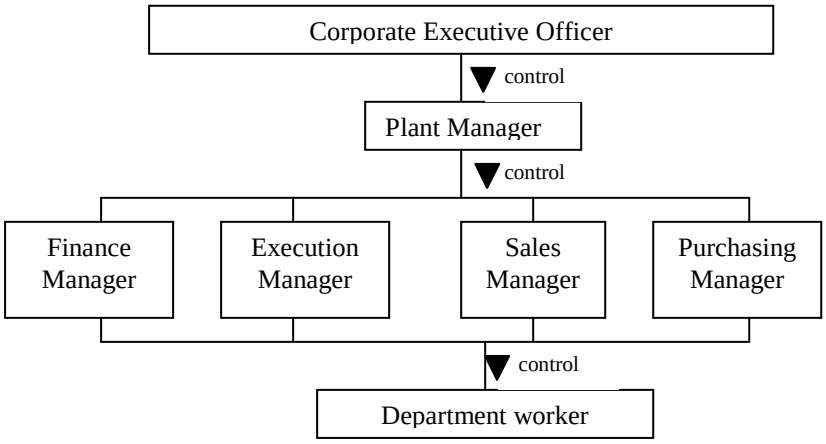


Figure 4.50 ESM template for enterprise organization structure

ESM template for enterprise supplier information:

The ESM template for enterprise supplier information is depicted in Figure 4.51. There are various types of information regarding a supplier for an enterprise, which are considered while selecting the supplier. They are the supplier's reliability, quality, client-base, financial stability, and after-sales support information.

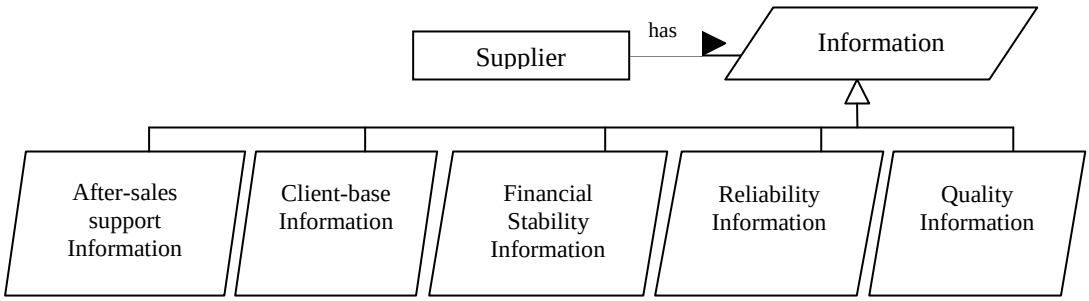


Figure 4.51 ESM template for supplier information

4.8.2 ESM template for enterprise information systems

In this section, the ESM template is developed for enterprise information system aspects, such as their functional view, cost, software interface, hardware interface, operating system, database management system, and manufacturer's information.

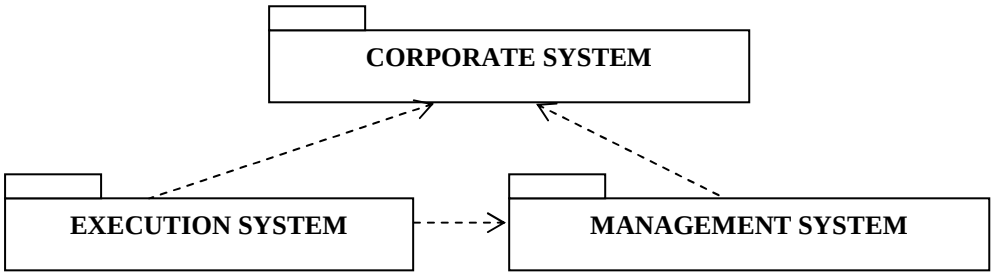


Figure 4.52 ESM template for information systems- functional view

ESM template for information systems- functional view:

The ESM template for the functional view of enterprise information systems is depicted in Figure 4.52 using 'package diagram' of UML. The enterprise has various information systems performing different functions (Section 4.5). These information systems can be categorized into three packages, corporate, execution and management systems. The execution systems depend on both the management and corporate systems, and the management systems depend on the corporate systems.

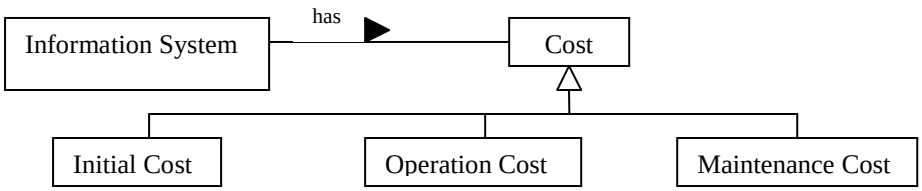


Figure 4.53 ESM for information system- cost

ESM template for information system- cost:

The ESM template for the cost of enterprise information systems is depicted in Figure 4.53. The information system of an enterprise has some costs, such as initial, operation, and maintenance costs, attached to it.

ESM template for information system- software interface:

The ESM template for the software interface of enterprise information systems is depicted in Figure 4.54. The enterprise information system has various types of software interfaces such as graphical user interface and non-graphical user interface.

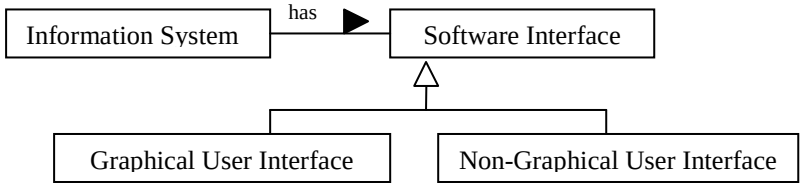


Figure 4.54 ESM for information system- software interface

ESM template for information system- hardware interface:

The ESM template for the hardware interface of enterprise information systems is depicted in Figure 4.55. The hardware of an enterprise has interfaces, such as control interface and non-control interface.

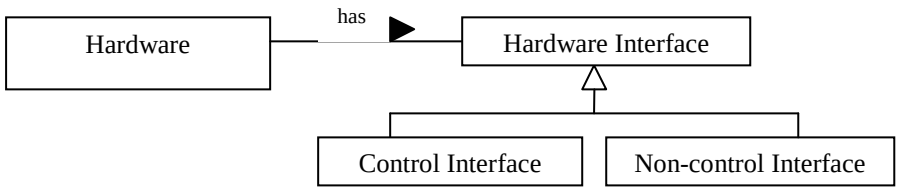


Figure 4.55 ESM template for information system- hardware interface

ESM template for information system- operating system (OS):

The ESM template for the OS of enterprise information systems is depicted in Figure 4.56. The enterprise information system has various types of OS, such as batch, time-shared, and real-time OS.

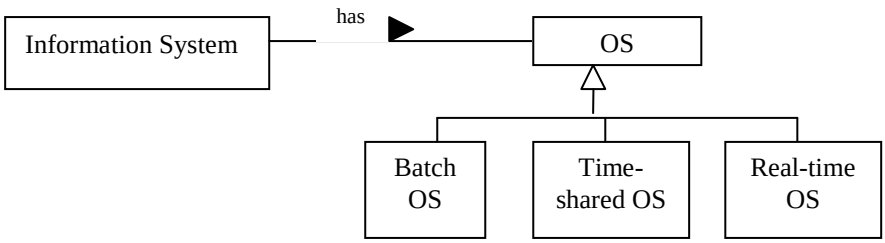


Figure 4.56 ESM template for information system- OS

ESM template for information system- database management system (DBMS):

The ESM template for the DBMS of enterprise information systems is depicted in Figure 4.57. The enterprise information system uses various types of DBMS, such as object-oriented (OO), extended relational, relational, and hierarchical DBMS.

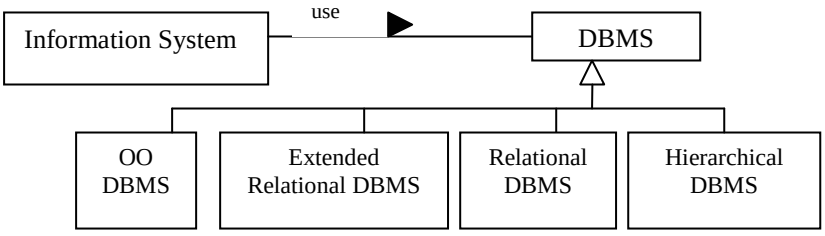


Figure 4.57 ESM template for information system- DBMS

ESM template for information system- manufacturer's information:

The ESM template for the manufacturer's information of enterprise information systems is depicted in Figure 4.58. The information system manufacturer of an

enterprise has various information, such as hardware, functionality, operating system, interface, and DBMS information.

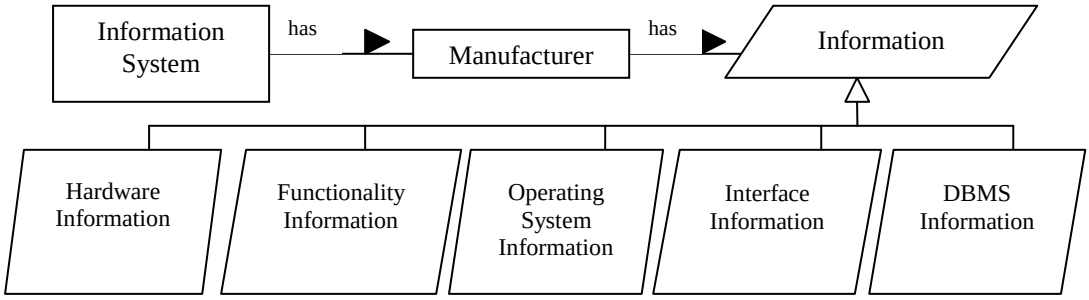


Figure 4.58 ESM template for information system- manufacturer's information

4.9 Conclusion

In this Chapter, the philosophical ground of the semantic model approach to EAI is elaborated. A fundamental notion of the ESM framework, ontology, is also discussed. Ontology modeling is done for both enterprise functions and enterprise information systems. Using this generic ontology, a systematic procedure for developing ontology for a particular type of enterprise, process enterprise, is detailed. Further, the procedure for developing ontology for a particular type of process enterprise, steel enterprise, from the ontology of generic process enterprise is described. Finally, the enterprise semantic model template is developed by using the ontologies developed in the previous sections. In the next Chapter, the procedure for instantiating the ESM template will be explained.

CHAPTER 5

A CASE STUDY

5.1 Introduction

This chapter demonstrates how the enterprise semantic model (ESM) framework is applied to develop the semantic model for a particular enterprise. The enterprise is called ABC, which is largely based on a real company, but its name is not disclosed. In Section 5.2, ABC is introduced. It is noted that in Chapter 4, a five-layer organizational architecture of the ESM framework was presented (see Figure 4.3). Part of the templates at level 4 will also be described in this chapter because they depend on a particular company situation; as such, those company specific templates are described in Section 5.3. A concrete semantic model is the result of instantiation of the templates; Section 5.4 presents such instantiation. Section 5.5 gives examples to show the potential of using the semantic model proposed in this thesis to address unsolved problems with those other approaches to EAI (see Chapter 1).

5.2 ABC Company

The enterprise for which the ESM is developed in this chapter will be called ABC. The enterprise's original name is not disclosed for reasons of confidentiality.

ABC’s sole line of business is steel making and fabricating. ABC is one of the leading suppliers of the wide and thick carbon hot rolled coil and discrete plate in North America. It is a major player in certain special steel markets, especially tubular products and flat-rolled alloy steels. ABC makes steel pipes ranging from 2-80" in diameter. ABC also performs cut-to-length operations on steel pipes. ABC has an annual capacity of 1,000,000 tons of steel produced in two electric arc furnaces. The major raw material used in the steel making process is ferrous scrap. ABC’s total annual consumption of ferrous scrap can be more than 1.1 million tons; thus, the company is a large recycler of steel. ABC includes a modern slab caster which processes liquid steel into continuously cast solid steel slabs up to 8" thick and up to 86" wide. The slab is then usually cut into 30' lengths, reheated in a gas-fired furnace to a temperature of 2300 degrees Fahrenheit, and rolled on a hot rolling mill into strips of steel from 0.090" to .750" thick, 36" to 77" wide and up to 3000' long which are then coiled up for ease in handling and transport. The rolling mill also produces hot rolled discrete plate from .500" to 2.5" thick and 48" to 72" wide. ABC purchases scrap from a network of scrap dealers and processors.

The electric arc steel making process uses electrical energy, which flows through graphite electrodes positioned above the raw materials so as to create an electrical arc reaching temperatures up to 5500 degrees Fahrenheit. The use of this form of energy makes ABC a large consumer of electricity. In addition, the graphite electrodes, which carry the electrical energy to the scrap and are slowly consumed in the process, are a source of carbon. A few of the other raw materials, including alloys such as manganese and silicon, are added to certain types of steel in order to impart special properties such as strength and corrosion resistance characteristics. Oxygen is used to remove impurities during the steel making process and to provide additional energy for melting the raw materials. Carbon dioxide and argon gases are used to shield the liquid steel from ambient air contamination during refining and pouring. Electric arc furnace steel making is environmentally friendly. Up to 46 million Imperial gallons of water are circulated daily in the steel melting and

casting operations, chiefly as a process coolant. This water is constantly re-treated, purified, and then recycled.

ABC employs directly and through its subsidiary companies more than 2,000 people. ABC, like other steel companies, has invested millions of dollars to make steel making more environmentally friendly. All of ABC's operating facilities have received ISO 14001 certification of their environmental management systems. The by-products of steel making are converted into other useful products. So far ABC has the following enterprise information systems: SAP™ ERP and DLGL™ systems for accounting and human resource management; SDC™ system for maintenance scheduling; Preactor™ for production scheduling; AutoCAD™ R12 for engineering; Outbound™ for shipping; Kinectrics™ for quality management; and Simens™ and Universal™ for process control.

5.3 Enterprise Semantic Model Template Specialized for ABC

The ESM developed in this section is specialized for ABC. From this ESM template, the semantic model for ABC can be developed through the process of instantiation. These templates are built upon the ontology developed in Chapter 4.

5.3.1 ESM template for enterprise functions specialized for ABC

In this section, the ESM template is developed for enterprise functional aspects such as business processes, enterprise resources, organization structure, and supplier information.

ESM template for manufacturing (business process):

The ESM template for a particular business process, manufacturing, is modeled in Figure 5.1. Manufacturing proceeds in the order from charging, melting, casting,

transportation, and finishing. All along, the manufacturing process sends information to various departments simultaneously.

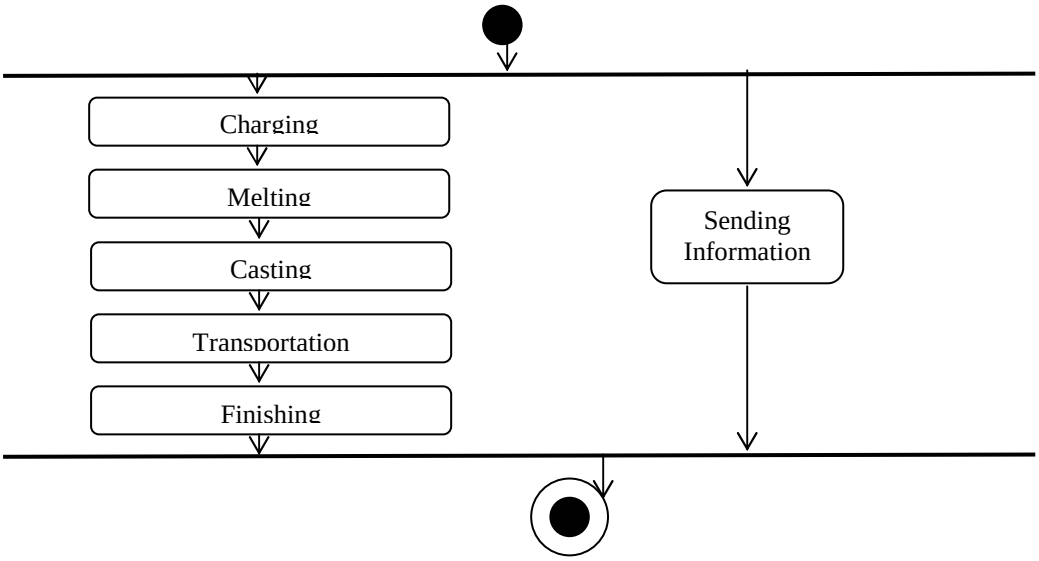


Figure 5.1 ESM template for manufacturing

ESM template for manufacturing equipment (resource):

The ESM template for a particular resource, manufacturing equipment, is modeled in Figure 5.2. The various manufacturing equipment are: container, furnace, processing equipment, and material handling equipment.

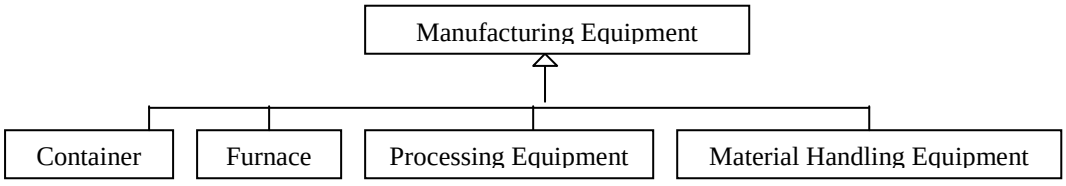


Figure 5.2 ESM template for manufacturing equipment

ESM template for organization structure:

The ESM template for organization structure specialized for ABC is modeled in Figure 5.3. The corporate executive officer controls the plant manager, who in turn

controls the direct production manager and support manager. These managers control their respective department workers.

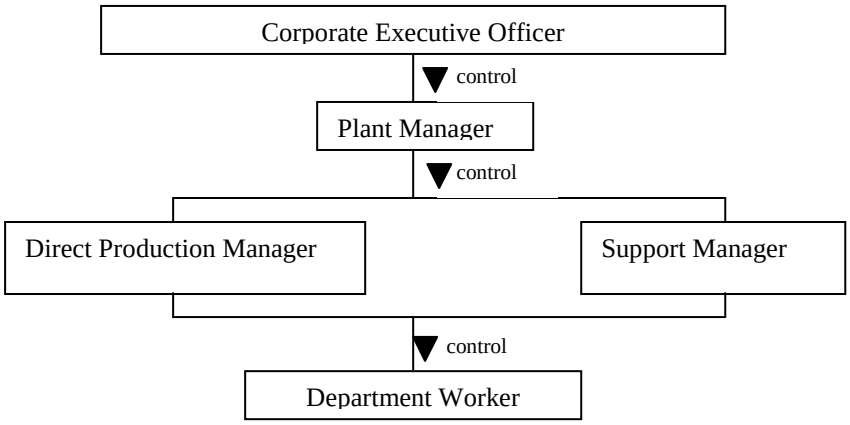


Figure 5.3 ESM template for organization structure

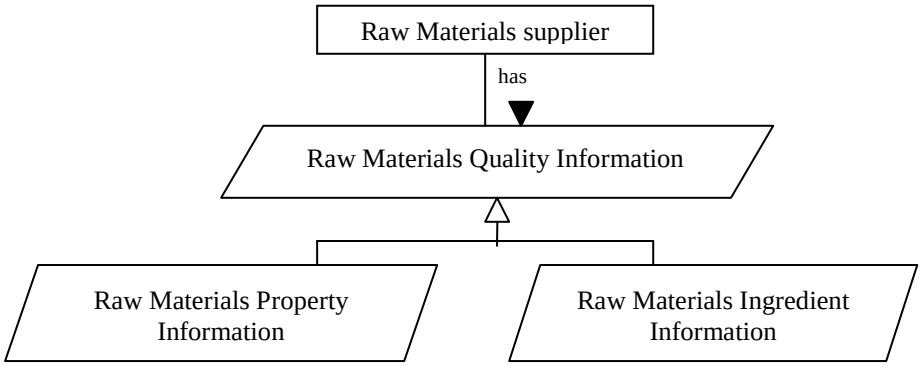


Figure 5.4 ESM template for supplier's information

ESM template for supplier's information:

The ESM template for a particular supplier's information, raw materials quality information, is modeled in Figure 5.4. The raw materials supplier has much information to be examined. For example, the various types of quality information to be considered when buying raw materials from the supplier include raw materials property information and raw materials ingredient information.

5.3.2 ESM template for enterprise information systems specialized for ABC

In this section, the ESM template is developed for the functional view of enterprise information systems.

ESM template for information systems- functional view:

The ESM template for functional view of information systems, specialized for ABC, is modeled in Figure 5.5. Engineering systems depend on corporate systems; execution systems depend on engineering systems; and process control systems depend on execution systems. The relationships between the remaining systems are explained in Section 4.8.

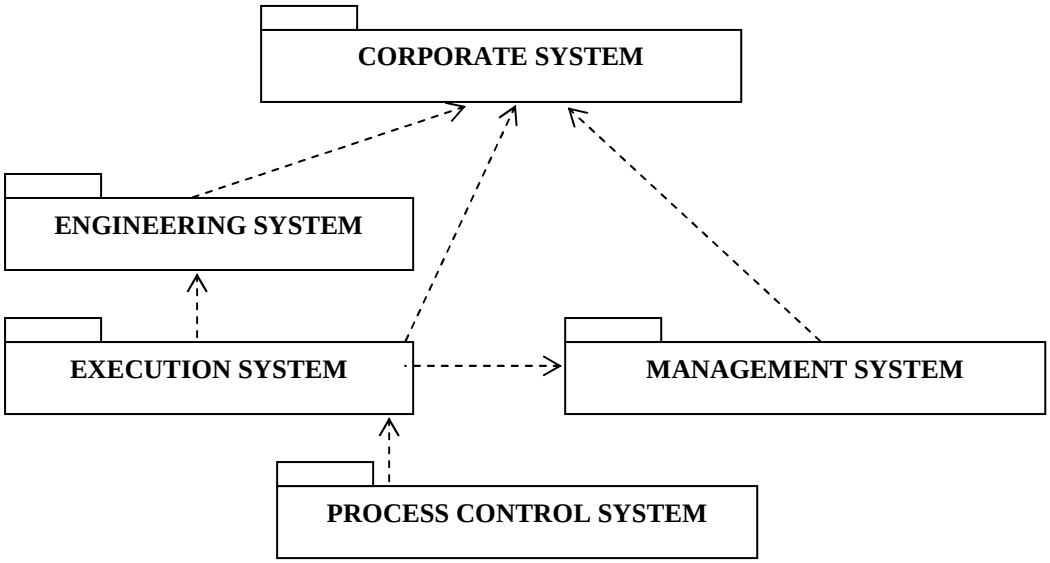


Figure 5.5 ESM template for information systems- functional view

5.4 Instantiation of Enterprise Semantic Model Template for ABC

The enterprise semantic model (ESM) for both the functions and information systems of ABC is developed by instantiating the ESM template developed in

Section 5.3. It is noted that the instantiated ESM developed in this section could go one level further. However, the instantiation is not undertaken to that level because the problem called “one class one instance or few instances” could occur [Zhang & Werff 1993], which has by itself caused much debate in the database community. In the following discussion, this will be noted.

5.4.1 ESM for ABC functions

In this section, the ESM for functional aspects of ABC such as manufacturing business process, equipment resources, organization structure and supplier information are modeled.

ESM for ABC business process- manufacturing:

The ESM for the manufacturing business process of ABC is modeled in Figure 5.6 by instantiating the ESM template shown in Figure 5.1. The charging includes bucket makeup and furnace charging. The melting includes melting in electric arc furnace, refining, tapping into ladle, adding alloy and de-sulphurizing agents, and slag removal. The casting includes casting in continuous slab caster and the transportation includes slab transportation by over-head crane. The finishing includes re-heating, rolling in 2-hi slabbing mill, shearing in shearing mill, finishing in 4-hi rolling mill, and cooling in cooling bed. The sending information includes sending information to engineering, HRD, R&D, finance, and maintenance departments. By modeling all the activities shown in Figure 5.6 as classes, and then instantiating them, instantiation can be made to go one level further. For example, slag removal can be modeled as a class, and the exact slag removed, **sulphur**, can be instantiated as an instance or an object of that class.

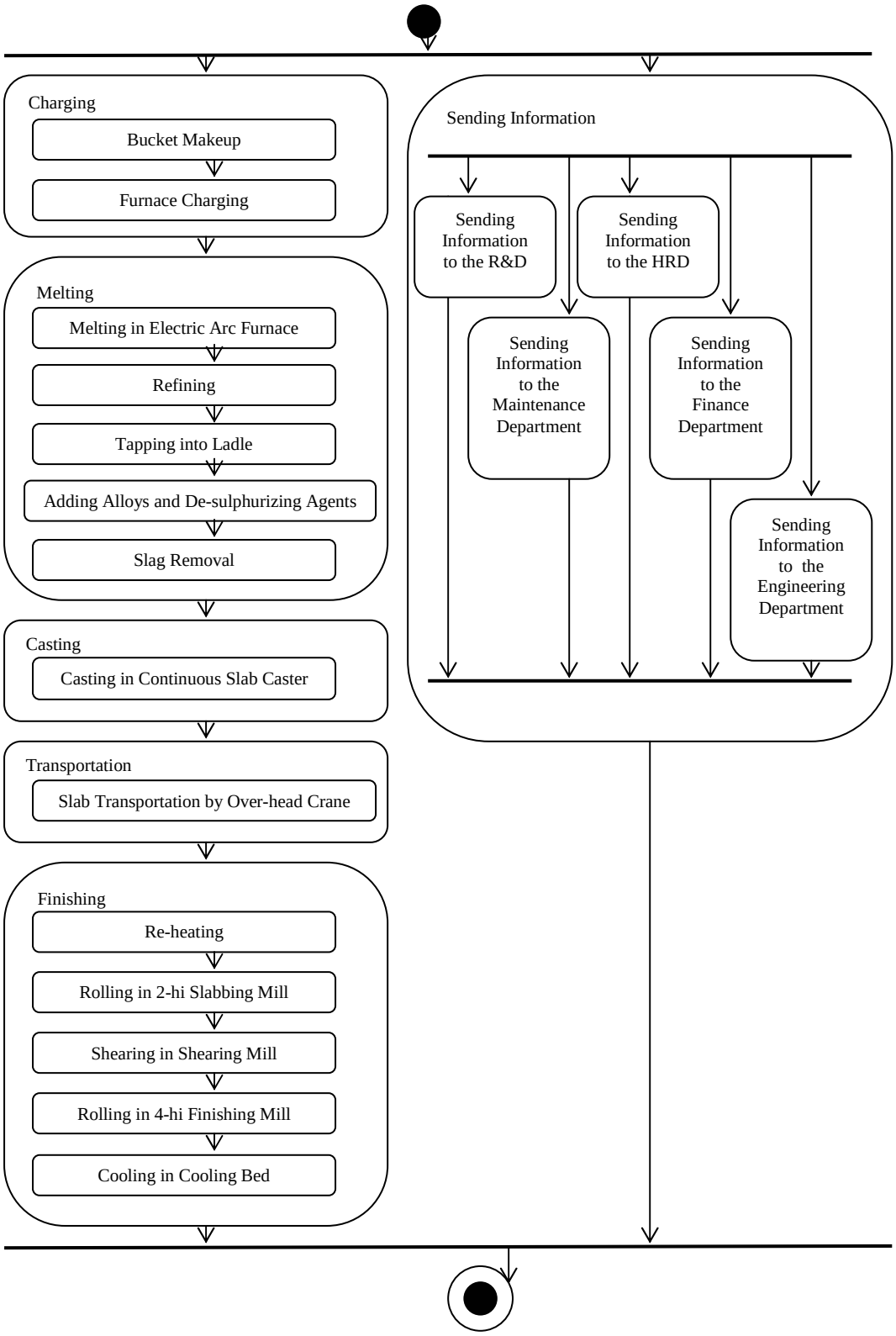


Figure 5.6 ESM for ABC business process- manufacturing

ESM for ABC resource- equipment:

The ESM for the manufacturing equipment of ABC is modeled in Figure 5.7, by instantiating the ESM template shown in Figure 5.2. Container includes ladle and bucket; furnace includes electric arc furnace, walking beam furnace, slab re-heat furnace, and ladle metallurgy furnace; processing equipment includes rolling mill, shearing mill, leveller, continuous slab caster, cooling bed, up-coiler, slabbing mill, and finishing mill; and material handling system includes transfer cars and over-head cranes. The classes shown in Figure 5.7 can be instantiated to one level further. For example, the class electric arc furnace can be instantiated to an object, **Fuchs**™, the exact electric arc type of furnace used in ABC.

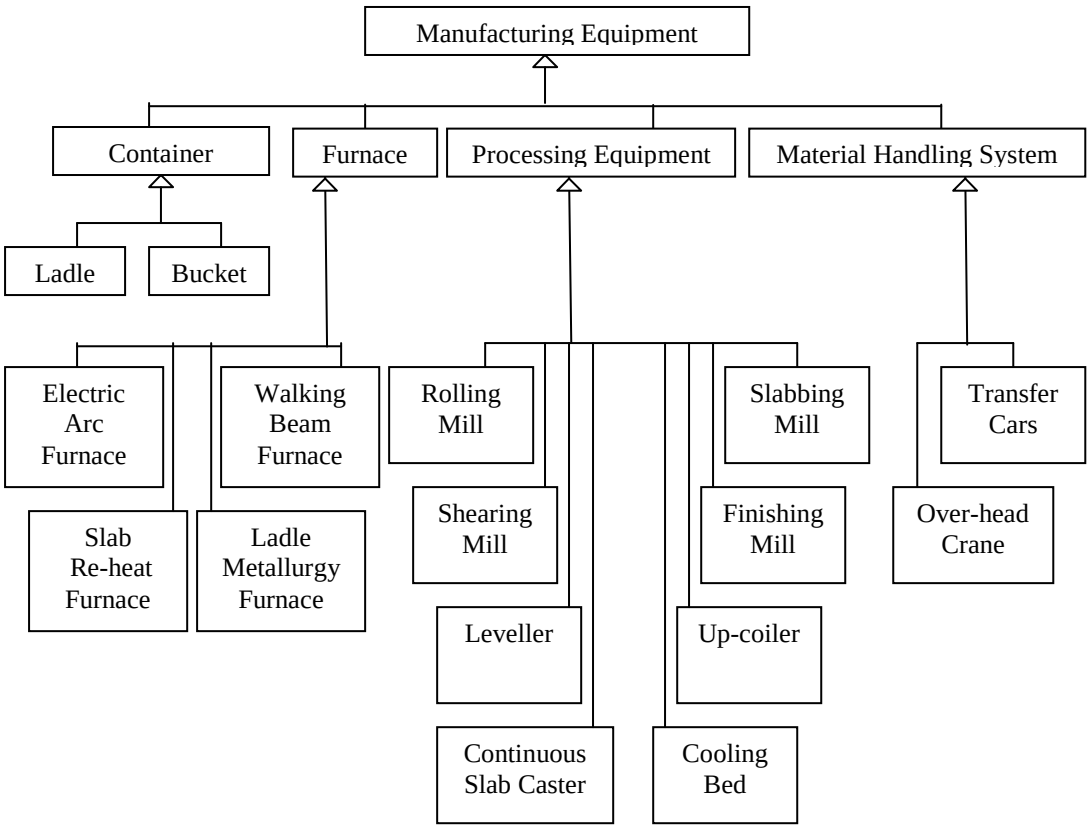


Figure 5.7 ESM for ABC resource- manufacturing equipment

ESM for organization structure of ABC:

The ESM for the organization structure of ABC is modeled in Figure 5.8, by instantiating the ESM template shown in Figure 5.3. Direct production manager includes purchasing, design, sales, manufacturing, inventory, and workflow managers; support manager includes finance, HRD, CRM, R&D, information systems, maintenance, quality control, safety control, shareholder relations, and environmental affairs managers; and department worker includes department supervisors and department laymen. The classes shown in Figure 5.8 can be instantiated to one level further. For example, the class sales manager can be instantiated to an object, the **exact name** of the sales manager.

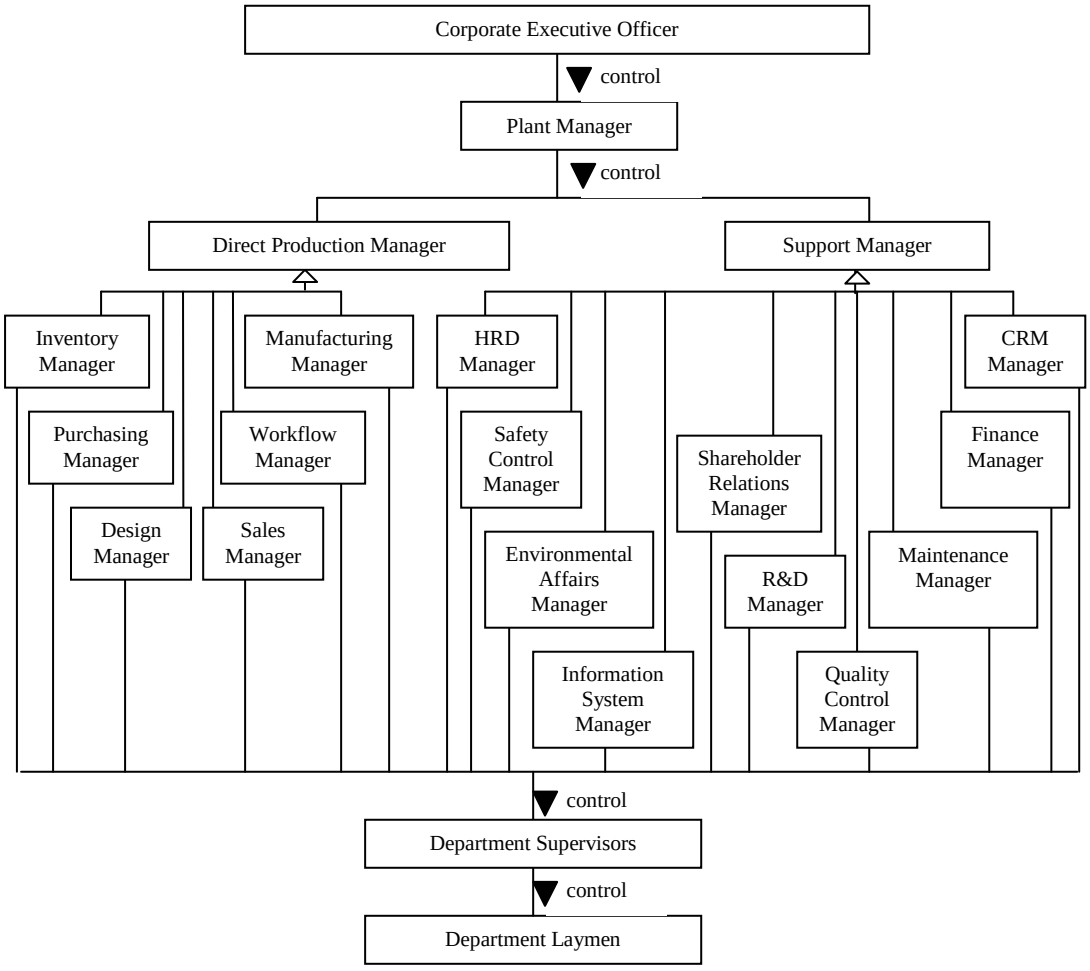


Figure 5.8 ESM for organization structure of ABC

ESM for supplier information of ABC:

The ESM for the supplier information of ABC is modeled in Figure 5.9 by instantiating the ESM template shown in Figure 5.4. Raw materials property information includes heat property, specific gravity, reactivity, and solubility information; and raw materials ingredient information includes mineralogical, chemical content, and hazardous ingredients information. The classes shown in Figure 5.9 can be instantiated to one level further. For example, the specific gravity information class can be instantiated to an object, the **exact specific gravity** of the raw material used in ABC.

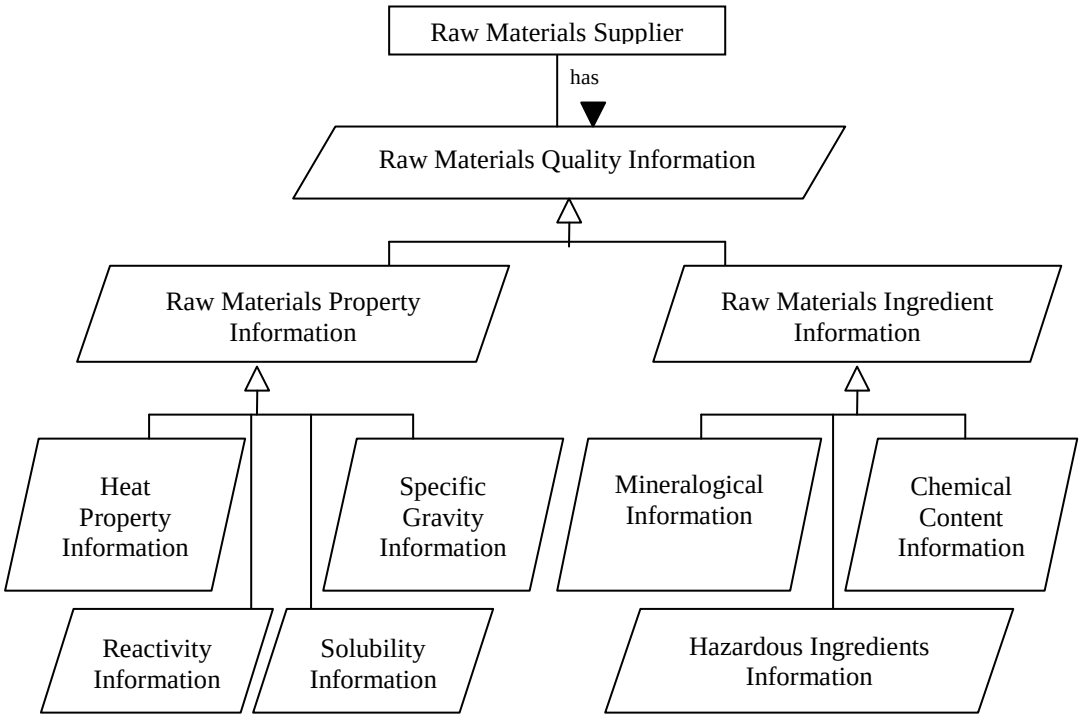


Figure 5.9 ESM for supplier information of ABC

5.4.2 ESM for ABC information systems

In this section, the ESM for the information systems aspects of ABC, such as their functional view, cost, software interface, hardware interface, operating system, database management system, and manufacturer's information, are modeled.

ESM for ABC information systems- functional view:

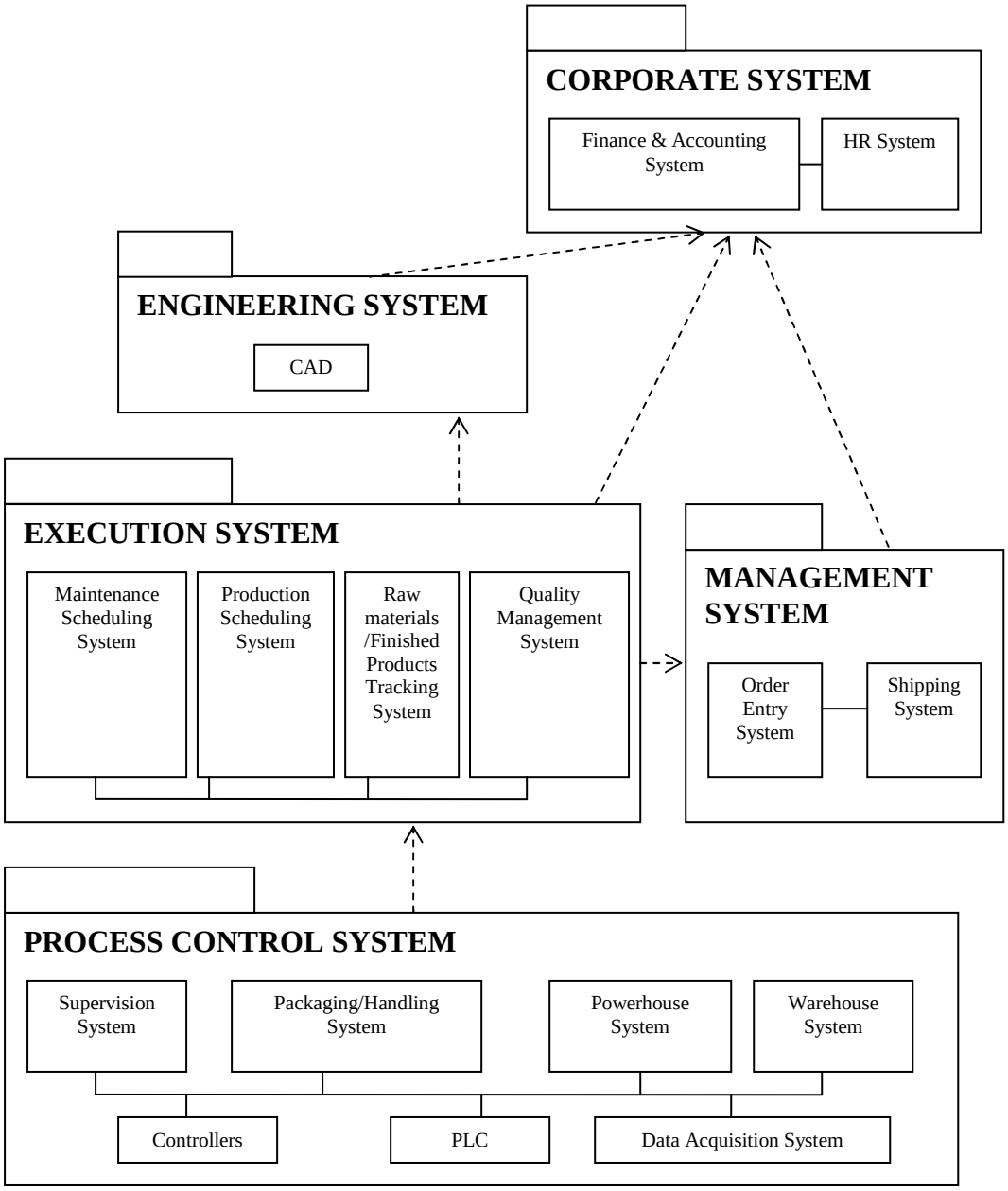


Figure 5.10 ESM for ABC information systems- functional view

The ESM for the functional view of ABC information systems is modeled in Figure 5.10, by instantiating the ESM template shown in Figure 5.5. The corporate system includes finance and accounting systems and human resource systems. The

engineering system includes CAD systems. The management system includes order entry and shipping systems. The execution system includes maintenance scheduling, production scheduling, raw materials and finished products tracking, and quality management systems. The process control system includes supervision, packaging/handling, powerhouse, warehouse, and data acquisition systems and also PLCs (Programmable Logic Controllers) and other controllers. The classes shown in Figure 5.10 can be instantiated to one level further. For example, the class CAD system can be instantiated to an object, **AutoCAD™ R12**, used in ABC.

ESM for ABC Information system- cost:

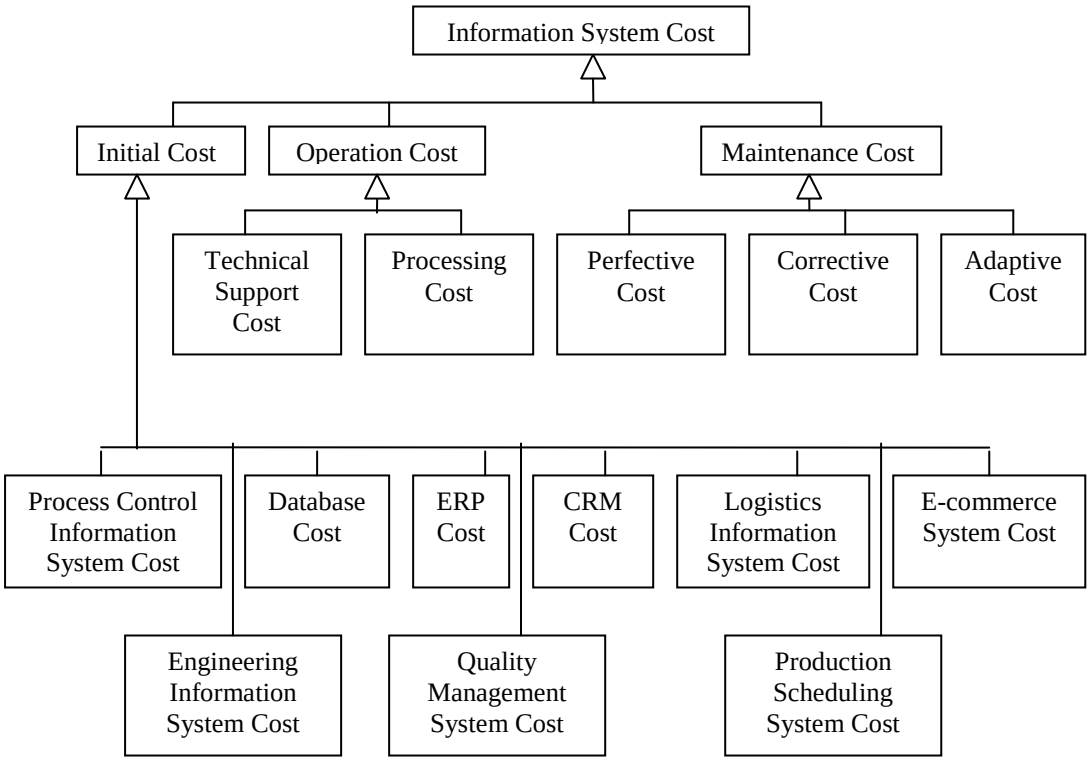


Figure 5.11 ESM for ABC information system- cost

The ESM for the cost of ABC information systems is modeled in Figure 5.11, by instantiating the ESM template shown in Figure 4.53. Initial cost includes process control information system, ERP, CRM, logistics, e-commerce, engineering, quality

management, production scheduling systems and database cost; operation cost includes technical support and processing cost; and maintenance cost includes perfective cost, corrective cost, and adaptive cost. The classes shown in Figure 5.11 can be instantiated to one level further. For example, the class ERP (Enterprise Resource Planning) cost can be instantiated to an object, the **exact cost of SAP™**, the ERP system used in ABC.

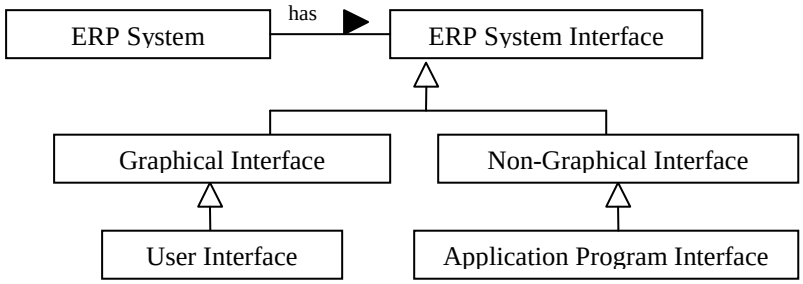


Figure 5.12 ESM for ABC information system- software interface

ESM for ABC information system- software interface:

The ESM for the software interface of ABC information systems is modeled in Figure 5.12, by instantiating the ESM template shown in Figure 4.54. ERP system interface includes graphical and non-graphical user interface. Graphical user interface includes user interface; and non-graphical interface includes application program interface (API). The classes shown in Figure 5.12 can be instantiated to one level further. For example, the class application program interface (API) can be instantiated to an object, **SAP™ Netweaver API**, the exact API used by the SAP™-ERP system in ABC.

ESM for ABC information system- hardware interface:

The ESM for the hardware interface of ABC information systems is modeled in Figure 5.13, by instantiating the ESM template shown in Figure 4.55. The process control system interface includes control and non-control interfaces. Control

interface includes user-control, motion control, safety system, sensor system, and I/O hardware system interfaces; and non-control interface includes weighing system and data analysis interfaces. The classes shown in Figure 5.13 can be instantiated to one level further. For example, the class motion control interface can be instantiated to an object, **RS 485**, the exact motion control serial of Universal™ Process Control system (**UPS 03**) used in ABC.

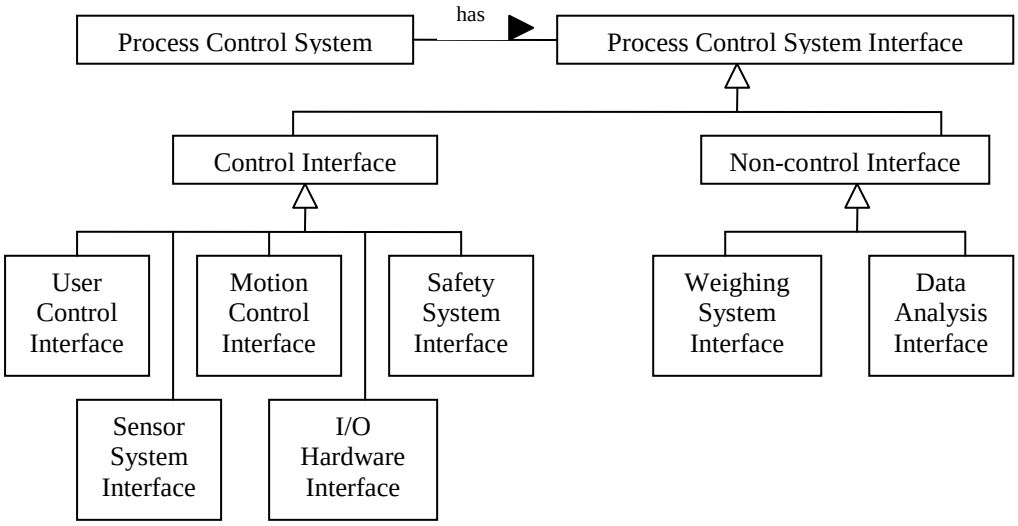


Figure 5.13 ESM for ABC information system- hardware interface

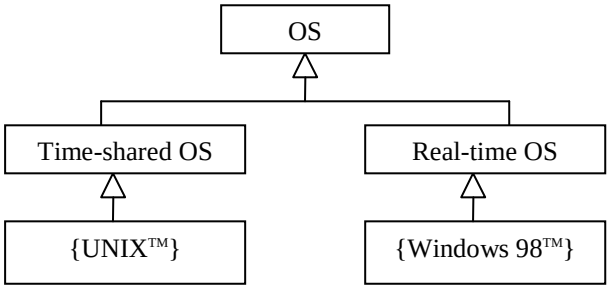


Figure 5.14 ESM for ABC information system- OS

ESM for ABC information system- operating system (OS):

The ESM for the operating system of ABC information systems is modeled in Figure 5.14, by instantiating the ESM template shown in Figure 4.56. **UNIX**™ is an instance of time-shared OS class and **Windows 98**™ is an instance of real-time OS class.

ESM for ABC information system- database management system (DBMS):

The ESM for the DBMS of ABC information systems is modeled in Figure 5.15, by instantiating the ESM template shown in Figure 4.57. **Oracle**™ and **DB2**™ are instances of relational DBMS class.

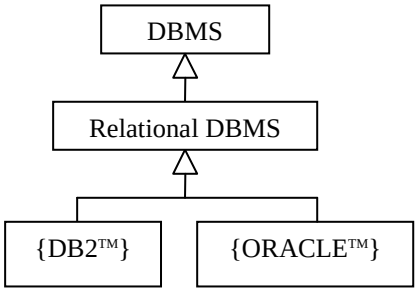


Figure 5.15 ESM for ABC information system- DBMS

ESM for ABC information system- manufacturer's information:

The ESM for the manufacturer's information of ABC information systems is modeled in Figure 5.16, by instantiating the ESM template shown in Figure 4.58. ABC's information system manufacturer uses ORACLE™ DBMS. Compatibility, capability, usability, security, version, reliability, interface, and tools information are the information needed while modeling the ORACLE™ DBMS information of the information system manufacturer. The classes shown in Figure 5.16 can be instantiated to one level further. For example, the class version information can be

instantiated to an object, **8i**, the exact version of **ORACLE™ DBMS** system used in ABC.

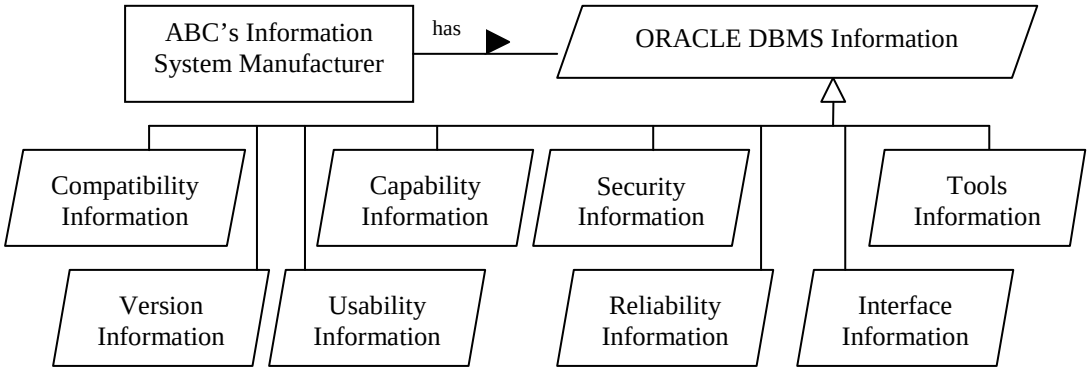


Figure 5.16 ESM for ABC information system- manufacturer's information

5.5 Examples

The situation where ABC wants to purchase a new CAD (Computer Aided Design) system, say SolidWorks™, is considered as an example. The top decision maker needs to ask those questions raised in Chapter 1. Some of them are as follows:

(i) Does this new CAD system create communication problems with the existing information systems in ABC?

(ii) Are any of the functions performed by this new CAD system already performed by systems existing in ABC?

(iii) Are any of the functions performed by this new CAD system not needed by ABC?

(iv) Does ABC have the required resources and organization structure to operate this new CAD system to its full functionality?

(v) Is it a wise financial decision to buy this CAD system?

(vi) If there is a problem with any of the questions mentioned above, when will the problem be resolved, i.e., when will ABC be able to purchase the new CAD system?

It is noticed that the following types of information have been seen in the ESM of ABC: (1) Business processes, (2) Resources, (3) Organization structure, (4) Functional view of information systems, (5) Software interface of information systems, (6) Operating systems of information systems, and (7) Database management systems of information systems. Furthermore, there is a knowledge-base about SolidWorks™. It is noted that the knowledge-base that has information about SolidWorks™ is not a part of the ESM of ABC, while it is a part of the knowledge-base for decision making for EAI. It is also noted that a particular ESM is just a storage of information about a particular company (e.g., ABC). The ESM becomes useful only when any decisions related to information systems support to the company is demanded; in this case, there will be an engine for making decisions based on general knowledge-bases. This decision making engine drives a reasoning process to lead to a decision. The engine is a piece of software, the same as a knowledge or an expert system known to the general artificial intelligence community. The development of this engine is out of the scope of this thesis, but the general knowledge-bases for the engine to work can be built using the templates developed in this thesis. The situation described above is shown in Figure 5.17. In Figure 5.17, demands can be initiated either from a need to enhance enterprise functions or a need to resolve conflicts between information systems.

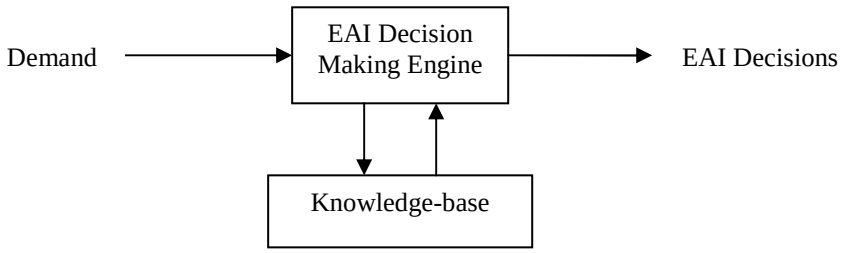


Figure 5.17 Decision making using knowledge-base

From the knowledge-base, we can retrieve the following information:

(1) SolidWorks™ is a CAD system

(2) SolidWorks™ has Microsoft .NET API™ software interface

(3) SolidWorks™ has Windows XP™ operating system

(4) SolidWorks™ has ORACLE™ database management system

(5) SolidWorks™ performs 2D and 3D modeling

(6) SolidWorks™ costs $6000 US, at present, but the price will be $4500 US, by 2004

(7) SolidWorks™ needs desktop computer resource and design personnel.

From the ESM of ABC, we can retrieve the following information:

(1) ABC has AutoCAD R12™, and SAP™ ERP to support CAD and ERP functions respectively

(2) SAP™ has SAP Netweaver API™ software interface

(3) SAP™ has Windows 98™ operating system

(4) SAP™ has ORACLE™ database management system

(5) AutoCAD™ performs only 2D modeling and ABC needs support for 3D modeling

(6) The funding available to buy a CAD software at present is $5000 US

(7) ABC possesses desktop computer resource and design personnel.

5.5.1 Identification of potential conflicts

It is derived from the ESM of ABC that function 2D modeling is presently supported by the CAD system currently used in ABC, AutoCAD R12™. Furthermore, it is derived from the ESM of ABC that function 3D modeling is not supported by AutoCAD R12™, and needs to be supported. From the knowledge-base, it is derived that SolidWorks™ supports both 2D modeling and 3D modeling. By comparing the functions supported by SolidWorks™, derived from the knowledge-base, and the functions performed by ABC, derived from the ESM of ABC, it is found that SolidWorks™ does not perform any function not needed by ABC. Hence, from the **functional support** viewpoint, there is only one conflict, "**functional redundancy"** (2D modeling).

It is found from the knowledge-base that to operate SolidWorks™, a desktop computer and design personnel is needed. From the ESM of ABC, it is derived that ABC possesses both desktop computer and design personnel. Hence, there is no conflict from the **resource and organization structure** viewpoint.

From the ESM of ABC, it is derived that SAP™ ERP, the ERP system used in ABC, needs to communicate with the SolidWorks™ CAD system. It is also derived from the ESM of ABC that the SAP™ ERP system uses SAP NetWeaver Application Program Interface™ (API). Furthermore, it is found from the knowledge-base that SolidWorks™ uses Microsoft .NET API™. In order to integrate SAP™ and SolidWorks™ from the **software interface** viewpoint, the compatibility issue between SAP NetWeaver API™ and Microsoft .Net API™ is reviewed from the knowledge base. According to the knowledge stored in the knowledge-base, SAP NetWeaver API™ and Microsoft .Net API™ are found to be compatible and, hence, there is no conflict from the software interface viewpoint.

From the ESM of ABC it is derived that the SAP™ ERP system supports the Windows 98™ **operating system** (OS). It is found from the knowledge-base that SolidWorks™ supports Windows XP™ OS. In order to integrate SAP™ and SolidWorks™ from the OS viewpoint, the compatibility issue between Windows 98™ and Windows XP™ is reviewed from the knowledge base. According to the knowledge stored in the knowledge-base, Windows 98™ and Windows XP™ are found to be compatible and, hence, there is no conflict from the OS viewpoint.

From the ESM of ABC, it is derived that the SAP™ ERP system uses ORACLE™ **database management system** (DBMS). It is found from the knowledge-base that SolidWorks™ also uses ORACLE™ DBMS. Hence SAP™ ERP and SolidWorks™ are compatible from the DBMS viewpoint.

From the ESM of ABC, it is derived that $5000 US is available for purchasing a new CAD system. It is found from the knowledge-base that the cost of SolidWorks™ is $6000 US. Hence, there is a conflict in the **cost**.

From the knowledge-base it is found that the price of SolidWorks™ will be reduced to $4500 US by January 2004. Hence, if ABC purchases SolidWorks™ by 2004, there will not be any financial conflict.

5.5.2 Decision-making

The following information derived from the previous section is useful in the decision making process:

(i) SolidWorks™ has no compatibility problems with the existing software in ABC that needs to communicate with it.

(ii) There is a functional redundancy problem, as both AutoCAD R12™ and SolidWorks™ perform 2D modeling.

(iii) SolidWorks™ does not perform any function not needed by ABC.

(iv) ABC has all the resources and organization structure needed to operate SolidWorks™.

(v) ABC does not have the funding to purchase SolidWorks™ at present.

(vi) By 2004, ABC will be able to purchase SolidWorks™.

The decision to buy SolidWorks™ by 2004 is made, as there is no other conflict except for one functional redundancy. However, the advantages of having SolidWorks™ are greater when compared to the redundancy problem and, hence, the decision to purchase SolidWorks™ by 2004 is made.

On a final remark, the above mentioned reasoning and decision processes can be largely automated. This is one of the key reasons that ESM has a high potential to ease the complexity and difficulty in dealing with EAI problems.

CHAPTER 6

CONCLUSION

6.1 Overview of the Thesis

This thesis started with a discussion of the need to investigate a methodology for enterprise application integration. Enterprise applications are software systems that deal with any information related to an enterprise. Examples of enterprise application are ERP (Enterprise Resource Planning) software and CAD (Computer Aided Design) software. Enterprise application integration (EAI) is a part of enterprise integration (EI). EAI is going to be very important because of the explosive expansion of information and rapid development of its processing methods. Existing studies on EAI reported in the literature are indeed numerous, but they have not satisfied industrial needs based on the researcher's industrial visiting experiences and academic analysis. While complaints from industry about the lack of methods of coping with EAI are usually in practical terms, the researcher's analysis has shown the following problems responsible for this situation. *First*, existing studies are focused on the modeling of enterprises only and not on enterprise applications (or software systems) that enterprises use. *Second*, the existing studies are too theoretical to be applied to practical questions constantly puzzling industrial managers' minds. These questions are, for example, (1) What are the conflicts, if any, that will be created with existing enterprise systems if a new software system called X is introduced? (2) What will be the consequence if

the new system to be introduced is changed after two years? (3) Why should the enterprise buy a new system, X, as it has a system called Y which works very well?

The goal of this thesis was set to provide a more effective solution to the enterprise application integration (EAI) problem. The following objectives were set up to achieve this goal.

Objective 1: *To justify further the semantic modeling approach for the EAI problem through an analysis of the need of enterprise integration or enterprise application integration and the shortcomings of the existing approaches.*

Objective 2: *To formulate an enterprise semantic model framework. The framework needs to demonstrate its genericity and its capability of solving the problems with existing approaches.*

Objective 3: *To develop a showcase for an enterprise semantic model and to demonstrate the effectiveness of the enterprise semantic model as a potential solution to the EI or EAI problem.*

A comprehensive and critical discussion on existing studies on EAI was presented in Chapter 2. The literary sources are from both the software engineering community and the industrial engineering community. In the software engineering community, those most popular methods and systems helpful to address the EAI problem were examined; these include COBRA™, J2EE™, middleware, and adapters. In industrial engineering community, well-known schools of work, such as CIMOSA, PERA, and so on, were examined. The problems mentioned above were further confirmed.

Based on a critical review of existing works and analyses, an enterprise semantic model approach was elaborated and developed. The starting point of this approach is such that these identified problems were addressed. This was responsible for

some general ideas proposed for developing this approach. *First*, modeling of ontology for enterprise was considered as a foundation. Both enterprise functions and enterprise applications were modeled. UML was identified as a suitable modeling language and used for representing the ontology and the model elements throughout the thesis work. *Second*, the organizational structure for the model was devised to have five layers to achieve genericity of the model. These five layers include (taking a process industry as an example): (i) level 1: generic enterprise, (ii) level 2: generic process enterprise, (iii) level 3: steel process enterprise, (iv) level 4: model template, and (v) level 5: model (i.e., instantiated model template). *Third*, modeling of enterprise applications was considered equally important as modeling of enterprise functions. This allows *walking through* between enterprise function and enterprise application.

A company, ABC, was visited during the course of this thesis study and was taken as a case study. It has been demonstrated how a semantic model for this company can be formulated. Finally, an example was shown of how the EAI problem could potentially be solved with the proposed approach.

6.2 Main Conclusions of the Thesis

(1) Enterprise application integration is becoming more and more difficult because of competition in the enterprise applications market, which is leading to a situation where it is impossible to have one unified operating system or language.

(2) Existing studies have not provided an effective solution to the enterprise application integration problem. The main shortcoming in the existing approaches is a lack of a powerful principle or concept for the integration of enterprise applications.

(3) The proposed concept called the enterprise semantic model appears to be effective to the enterprise application integration problem. This is attributed to several novel ideas behind this semantic model: (a) the five layer organizational

structure, (b) the modeling of enterprise application ontology and its relationship with enterprise function ontology, and (c) the relationship between the framework and the model.

6.3 Contributions of the Thesis

The main contributions of the thesis are: (i) identification of the bottle neck problems with existing studies and practices of EAI, and (ii) development of a new approach called the semantic model approach.

The semantic model approach contains the following salient points: (1) modeling of enterprise ontology using UML, (2) organization of enterprise ontology in terms of the data abstraction: generalization/specialization, (3) explication of relationship between ontology and data/database model, and (4) integrated modeling of both enterprise function and enterprise application.

One of the important contributions in intelligent and knowledge systems is the demonstration of the path from ontology to a concrete model. Other approaches, for example the TOVE enterprise ontology approach, are only on the fundamental level, i.e., on the level of a particular manufacturing process, say design, or at the data/ database modeling level for a specific database design where the semantics of concepts used in a data/database are left in the minds of modellers.

6.4 Future Work

There are still several issues regarding enterprise applications integration that have not been addressed, and they need to be studied further.

The first issue concerns the ways of predicting the evolution of a particular piece of software or paradigm. This issue is crucial to achieving an optimum trade-off for a

particular company between short-term gain and long-term suitability. A company could invest in an enterprise application software system today in order to obtain better support to its existing business functions, but software has a life and can become obsolete, say in two years. The obsolesce of this particular software may bring some constraints for introducing other new software systems due to their inability to communicate with the existing software. This issue could also be viewed in another way: How does one make decisions under uncertain situations (i.e., uncertainty in evolution of software)?

The second issue concerns the relationship between the enterprise semantic model and those tools that are available, e.g., adapters, middleware, etc. This issue is important as the development of these tools will certainly have an impact on the development of enterprise applications. An idea for this issue can be outlined as follows. One can develop a knowledge base using the enterprise application ontology and template developed in this thesis. The knowledge base will contain all the marketed middleware and adapters. Then, when one develops a solution to a particular EAI problem, the whole solution package will also include a specification of these tools.

The third issue is how to implement the ESM in general. Software of ESM is not a program like enterprise resource planning (ERP) or database management system. ESM will be implemented as a database application system, which means that (1) ESM will be built upon a database or knowledge management system, e.g., ORACLE™. (2) There will be a program system to perform a general process management task. In the ESM database system, the data dictionary will contain all the definitions of ontology and templates developed in preceding chapters. A particular enterprise will be treated as instances of these templates. The implementation of the data dictionary will be incrementally carried out, as the ESM model is developed as having the property of extendibility through its generalization/specialization architecture. The incremental implementation strategy is as follows. Suppose that implementation for an enterprise A is done, for which

ontology is implemented and denoted as O_A. When enterprise B is implemented, ontology O_A is first reviewed and extended (if needed) for enterprise B. This extension of ontology is denoted as O_B. As such, after both enterprises are implemented, ontology of ESM has been extended to $O_A + O_{B.}$ In this way, with more implementations for enterprises, the completion of ESM can be gradually achieved.

The process management program system as mentioned can be made fairly straightforwardly. In this system there is a decision making system which is like an expert or knowledge system. The expert system uses a knowledge-base which includes knowledge about the commercial enterprise application programs. Obviously, this knowledge-base can take the ontology and template (defined for the information system of an enterprise) developed in preceding chapters.

REFERENCES

[Baan 2003] Baan. (2003). *iBaan Enterprise- Solution for a lean operational environment.* Retrieved March 25, 2003 from the Baan Web site: http://www.baan.com/solutions/enterprise/

[Berio & Vernadat 1999] Berio, G., Vernadat, F.B. (1999). New Developments in Enterprise Modeling using CIMOSA. *Computers in Industry, 40* (2), pp. 99-114. Elsevier Science.

[BizTalk 2002] Microsoft BizTalk. (2002). *Microsoft BizTalk server: Product overview*. Retrieved October 27, 2002 from the Microsoft Website: http://www.microsoft.com/biztalk/evaluation/overview/biztalkserver.asp

[Booch et al. 1999] Booch, G., Rumbaugh, J., Jacobson, I. (1999). *The Unified Modeling language user guide*. The Addison-Wesley Object Technology Series. Addison-Wesley, U.S.A.

[Cadarette & Durward 2001] Cadarette, P., Durward, K.M. (2001). *Achieving a Complete Enterprise Integration Strategy.* Retrieved October 20, 2002 from the *ebizQ* Web site: http://eai.ebizq.net/str/cadarette_1.html

[Christensen et al. 1995] Christensen, L. C., Johansen, B. W., Midjo, N., Onarheim, J., Syvertsen, T. G., Totland,T. (1995). Enterprise Modeling – Practices and Perspectives. *Proceedings of ASME Ninth Engineering Database Symposium. Boston, U.S.A., September 1995.*

[CIMOSA 1996] CIMOSA Association. (1996). *CIMOSA: A Primer on key concepts, purpose and business value*. Retrieved August 27, 2002 from the CIMOSA Association Website: http://cimosa.cnt.pl/Docs/Primer/primer0.htm

[CORBA 2002] Object Management Group. (2002). *The Common Object Request Broker: Architecture and Specification.* Retrieved September 2, 2002 from the Object Management Group Web site: http://cgi.omg.org/docs/formal/02-06-01.pdf

[Cummings & Hanson 2002] Cummings, G., Hanson, K. (2002). Interaction Integration. *Enterprise Application Integration (EAI) journal, April 2002.* Thomas Communications Inc.

[Doumeingts & Chen 1996] Chen, D., Doumeingts, G. (1996). The GRAI-GIM reference model, architecture and methodology. In Bernus, P., Nemes, L., Williams, T. (Ed.), *Architecture for Enterprise integration*. Chapman & Hall.

[eJai 2001] Enterprise Java Application Integration (eJai). (2001). *eJai Universal Adapter Whitepaper*. Retrieved April 14, 2002 from the eJai Web site: http://www.igs.com/products/eJai%20Universal%20Adapter/eUA%20product%20sheet_final.pdf

[EJB 2002] Roman,E., Ambler,S., Jewell,T. (2002). *Mastering Enterprise Javabeans* ™. Wiley computing publishing, John Wiley & Sons, Inc., U.S.A.

[Eriksson & Penker 2000] Eriksson, H.E., Penker, M. (2000). *Business modeling with UML: business patterns at work*. John Wiley & Sons, New York, U.S.A.

[Fox 1993] Fox, M.S. (1993). Issues in Enterprise Modeling. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Le Touquet, France, October 1993*, pp. 86-92.

[Fox et al. 1993] Fox, M.S., Chionglo, J.F., Fadel, F.G. (1993). A Common-Sence Model of the Enterprise. *Proceedings of the 2nd Industrial Engineering Research Conference, Los Angeles, U.S.A.*, pp. 425-429.

[Fox & Gruninger 1994] Fox, M.S., Gruninger, M. (1994). The Role of Competency Questions in Enterprise Engineering. *Proceedings of the IFIP WG5.7 Workshop on Benchmarking- Theory and Practice, Trondheim, Norway, June 1994.*

[Fox & Gruninger 1998] Fox, M.S., Gruninger, M. (1998). Enterprise Modeling. *AI Magazine, 19 (3), pp. 109-121.* AAAI Press.

[Gosain & Thillairajah 2002] Gosain, S., Thillairajah, V. (2002). *EAI: The Business Drivers and Technical Challenges.* Retrieved September 12, 2002 from the *ebizQ* Web site: http://eai.ebizq.net/str/gosain_1a.html

[Green 1999] Green, S.D. (1999). The Dark Side of Lean Construction: Exploitation and Ideology. *Proceedings of the Seventh Annual Conference of the International Group for Lean Construction, Berkeley, California, USA, 26-28 July-1999.*

[i2 2003] i2. (2003). *Supply Chain Management.* Retrieved March 25, 2003 from the i2 Web site: http://www.i2.com/solutionareas/scm/index.cfm#

[Kosanke 96] Kosanke, K. (1996). Comparison of Enterprise Modelling Methodologies. *Proceedings of DIISM'96, Katsheuvel, Netherlands, September 15-18, 1996.* Chapman & Hall.

[Kramp & Coulson 2000] Kramp, T., Coulson, G. (2000). The Design of a flexible communications framework for next generation middleware. In Drew, P., Meersman, R., Tari, Z., Zicari, R. (Ed.), *DOA'00- International Symposium on*

Distributed Objects and Applications (pp. 273-283). IEEE Computer Society, California, U.S.A.

[Li et al. 1999] Li, Q., Tso, S.K., Zhang, W.J. (1999). Generalization of Strategies for Product Data Modeling with Special Reference to Instance-As-Type Problem. *Computers in Industry, 41,* pp. 25-34.

[Lim et al. 1997] Lim, S.H, Juster, N., De Pennington, A. (1997). Enterprise Modeling and Integration: A Taxonomy of seven key aspects. *Computers in Industry, 34 (3),* pp. 339-359. Elsevier Science

[Lin 1999] Lin,H., Fan,Y., Wu,C. (1999). The research of integrated enterprise modeling method based on workflow model. *7th International conference on ETFA, Barcelona, October 1999,* pp. 187-193.

[Linthicum 2001] Linthicum, D. (2001). *B2B Application Integration: e-business-enable your enterprise*. Addison-Wesley, U.S.A.

[Lutz 2000] Lutz, J.C. (2000). EAI Architecture patterns. *Enterprise Application Integration (EAI) journal, March 2000.* Thomas Communications Inc.

[Ma et al. 1999] Ma, Z.M., Zhang, W.J., Ma, W.Y. (1999). View Relation for Schema Integration on Multiple Databases and Data Dependencies. *Proceedings*

of 9th International Database Conference on Heterogeneous and Internet Database, Hong Kong, pp. 278-289.

[Migliore 2001] Migliore, M. (2001). *Hurwitz Study Reveals Integration a Growing Concern for Enterprises.* Retrieved December 20, 2002 from the Web Services report Web site: http://www.capeclear.com/news/pressroom/reports/web_services_report.htm

[McCoy et al. 2002] McCoy, D., Morello, D.T., Miklovic, D., Earley, A., Nicolett, M., Fulton, R., et al. (2002). *Gartner predicts 2002: Top 10 predictions.* Retrieved December 20, 2002 from the Gartner Web site: http://www3.gartner.com/DisplayDocument?doc_cd=103726

[Naiburg & Maksimchuk 2001] Naiburg, E.J., Maksimchuk, R.A. (2001). *UML for database design*. Addison-Wesley, U.S.A.

[Nayak et al. 2001] Nayak, N., Bhaskaran, K., Das, R. (2001). Virtual Enterprises: Building Blocks for Dynamic E-business. *Australian Computer Science Communications, Proceedings of the Workshop on Information Technology for Virtual Enterprises, Queensland, Australia, January-2001.*

[Oracle 2003] Oracle. (2003). *Oracle9i Database, Features.* Retrieved March 25, 2003 from the ORACLE Web site:

http://www.oracle.com/ip/deploy/database/oracle9i/index.html?oracle9idb_features2.html

[Presley et al. 2001] Presley, A., Sarkis, J., Barnett, W., Liles, D. (2001). Engineering the Virtual Enterprise: An Architecture-Driven Modeling Approach. *International Journal of Flexible Manufacturing Systems, 13 (2),* pp. 145- 162. Kluwer Academic Publishers.

[Rathwell 2001] Rathwell, G. (2001). *Introduction to PERA: Perdue Enterprise Reference Architecture.* Retrieved October 15, 2002 from the Perdue Enterprise Reference Architecture Web site: http://www.pera.net/

[Rational 2000] Rational. (2000). *The UML and data modeling.* Retrieved August 15, 2002 from the Rational Web site: http://www.rational.com/media/whitepapers/Tp180.PDF

[Smith & Smith 1977] Smith, J.M., Smith, D.C.P. (1977). Database abstraction: Aggregation and Generalization. *ACM Transactions on Database Systems, 2 (2),* pp. 105-133.

[Szegheo 2000] Szegheo, O. (2000). Introduction to Enterprise Modeling. In Rolstadås, A., Andersen, B. (Ed.), *Enterprise Modeling – Improving Global*

Industrial Competitiveness (pp. 21-33). Kluwer Academic Publishers, Boston, U.S.A.

[TechMetrix research 2002] TechMetrix research. (2002). *Which technology for tomorrow's EAI?* Retrieved August 24, 2002 from the *ebizQ* Web site: http://e-serv.ebizq.net/aps/techmetrix_2.html

[Ter Bekke 1992] Ter Bekke, J.H. (1992). *Semantic Data Modelling*. Printice Hall.

[Traverse 2001] Traverse, C. (2001). Adapting to Total Integration. *Enterprise Application Integration (EAI) journal, September 2001*. Thomas Communications Inc.

[Trujillo et al. 1997] Trujillo, J., King, G.A., Palomar, M. (1997). Semantic Data Modelling for databases: Issues of modelling and teaching the paradigm. *International Symposium on Software Engineering in Universities (ISSEU'97), Rovaniemi, Finland.*

[van Stekelenborg 1996] van Stekelenborg,R.H.A. (1996). On the Way to Supportive Information Technology for Contemporary Industrial Purchasing: A summary of four years of Dutch design-oriented research. *Heading for New Frontiers in Purchasing and Supply Management, Proceedings of the 5th*

International Annual IPSERA Conference, Eindhoven, NL, April, 1996, pp. 345-363.

[Warnecke & Huser 1995] Warnecke, H.J., Huser, M. (1995). Lean production. *International Journal of Production Economics, 41 (1-3),* pp. 37-43. Elsevier Science.

[Whitman & Huff 2001] Whitman, L., Huff, B. (2001). On the use of Enterprise Models. *International Journal of Flexible Manufacturing Systems, 13 (2),* pp. 195-208. Kluwer Academic Publishers.

[Williams 2000] Williams, T. (2000). *Workflow Management within the ARIS Framework.* Retrieved October 20, 2002 from the Architecture of Integrated Information System Web site: http://www.pera.net/Methodologies/ARIS/ARIS_Paper_by_Ted_Williams.html

[Yoshikawa et al. 1994] Yoshikawa, H., Tomiyama, T., Kiriyama, T., Umeda, Y. (1994). An Integrated Modeling Environment Using the Metamodel. *Annals of the CIRP, 43 (1),* pp. 121-124.

[Zhang et al. 1999] Zhang, W.J., Li, Q. (1999). Information Modelling for Made-to-Order Virtual Enterprise Manufacturing Systems. *International Journal of Computer Aided Design, 31,* pp. 611-619.

[Zhang & Werff 1993] Zhang, W., Werff, K van der. (1993). Guidelines for Product Data Model Formulation Using Database Technology. *Proceedings of International Conference on Engineering Design (ICED'93), Vol. 3, The Hague, The Netherlands,* pp. 1618-1626.

[Zhang & Werff 1994] Zhang, W., Werff, K van der. (1994). A critique of conceptual data modelling notions relative to the machine design domain. *Proceedings of 1994 ASME Engineering Database Symposium, USA,* pp. 59-66.

[Zrnec et al. 2001] Zrnec, A., Bajec, M., Krisper, M. (2001). Enterprise Modeling with UML. *Electro Technical Review, 68(2-3),* pp. 109-114.

APPENDIX A

ACTIVITY ONTOLOGY

Time and action ontology:

Context Situation inv:

$s_i < s_j$ implies $situation_i$ earlier than $situation_j$

Context Situation inv:

$s_j.(a_i.s_i)$ implies $situation_j$ results from performing $action_i$ in $situation_i$

Context Situation inv:

$s_i.t_i$ implies $situation_i$ starts in $time_i$

Context Duration inv:

$t_i < t_j$ implies $time_i$ earlier than $time_j$

Context Situation inv:

$s_i.t_i$ and $(s_j.(a_i.s_i)).t_j$ implies $t_i < t_j$

If $situation_i$ starts at $time_i$ and $situation_j$ resulting from performing $action_i$ in $situation_i$ starts at $time_j$, then it implies that $time_i$ is earlier than $time_j$.

Context State inv:

Commit

Pre: Status=Possible

Post: -

An action Commit takes place with the pre-condition that the status of the state is possible.

Context State inv:

Complete

Pre: (Status=Enabled) or (Status=Re-enabled)

Post: Status<>Enabled

An action Complete can take place with the pre-condition that the status of the state is either enabled or re-enabled and its post-condition is that status of the state is not enabled.

Context State inv:

Disenable

Pre: (Status=Enabled) or (Status=Re-enabled)

Post: -

An action Disenable can take place with the pre-condition that the status of the state is either enabled or re-enabled.

Context State inv:

Re-enable

Pre: Status=Disenabled

Post: -

An action Re-enable can take place with the pre-condition that the status of the state is disenabled.

Context State inv:

Duration= $t - t'$ implies $(a.S).t = Enable$ and $(a.S).t' = Complete$

Duration of the activity is $t - t'$, when an enable action takes place in state-S and time-t and disenable action takes place in state-S and time t'.

Activity and state ontology:

Ontology of status of state:

Context State inv:

Status=Committed implies $a.s_{i-1} = Commit$ and $a.s_i <> Enable$

The status of a state is committed in $situation_i$, if a commit action occurred in the preceding situation, $situation_{(i-1)}$ and an enable action did not occur.

Context State inv:

Status=Enabled implies $a.s_{i-1} = Enable$ and

$((a.s_i <> Complete) or (a.s_i <> Disenable))$

The status of a state is enabled in a $situation_i$ if an enable action occurred in the preceding situation, $situation_{(i-1)}$ and a complete action or disenable action did not occur.

Context State inv:

Status=Disenabled implies $a.s_{i-1} = Disenable$ and $a.s_i <> \text{Re}-enable$

The status of a state is disenabled in a $situation_i$ if a disenable action occurred in the preceding situation, $situation_{(i-1)}$ and a re-enable action did not occur.

Context State inv:

Status=Re-enabled implies $a.s_{i-1} = re-enable$ and

$((a.s_i <> Complete) or (a.s_i <> Disenable))$

The status of a state is re-enabled in a $situation_i$ if re-enable action occurred in the preceding situation, $situation_{(i-1)}$ and a complete action or disenable action did not occur.

Context State inv:

Status=Completed implies $a.s_{i-1} = Complete$ and $a.s_i <> Commit$

The status of a state is completed in a $situation_i$ if a complete action occurred in the preceding situation, $situation_{(i-1)}$ and a commit action did not occur.

Context State inv:

Status=Possible implies $a.s_{i-1} = Complete$ and $a.s_i <> Commit$

The status of a state is possible in a $situation_i$ if a complete action occurred in the preceding situation, $situation_{(i-1)}$ and a commit action did not occur.

Ontology of status of activity:

Context Activity inv:

ES implies enabling state

Context Activity inv:

CS implies caused state

Context Activity inv:

Status=Dormant implies ES=Committed and ES<> Enabled

The status of an activity is dormant if the status of its enabling state is committed and not enabled.

Context Activity inv:

Status=Executing implies ES=Enabled or CS=Enabled

The status of an activity is executing if the status of its enabling state or caused state is enabled.

Context Activity inv:

Status=Suspended implies ES=Disenabled or CS=Disenabled

The status of an activity is suspended if the status of its enabling state or caused state is disenabled.

Context Activity inv:

Status=Re-executing implies ES=Re-enabled or CS=Re-enabled

The status of an activity is re-executing if the status of its enabling state or caused state is re-enabled.

Context Activity inv:

Status=Terminated implies ES=Completed or CS=Completed

The status of an activity is terminated if the status of its enabling state or caused state is completed.

Non-terminal States Ontology:

Context State inv:

$a.(S.R)$ implies action occurring on Resource R in State S

Context State inv:

Disjunctive

Pre: $((a.(S.R_1)).s_i) = Enable$ or …or $((a.(S.R_n)).s_i) = Enable$

Post: Status <> Status@pre

The state is Disjunctive, if the status of the state can be changed, when one of the resources from R_1 to R_n has been selected and its status has been changed. The keyword Status@pre refers to the status before the action takes place.

Context State inv:

Conjunctive

Pre: $((a.(S.R_1)).s_i) = Enable$ and …and $((a.(S.R_n)).s_i) = Enable$

Post: Status <> Status@pre

The state is Conjunctive, if the status of the state can be changed, only when all of the resources from R_1 to R_n have been selected and its status has been changed.

Context State inv:

Exclusive

Pre: $((a.(S.R_i)).s_i) = Enable$ and $((a.(S.R_j)).s_i) <> Enable$

Post: Status <> Status@pre

The state is Exclusive, if the status of the state can be changed, only when one of the resources from R_i to R_j has been selected and its status has been changed.

Context State inv:

Not

Pre: $((a.(S.R_i)).s_i) <> Enable$

Post: Status <> Status@pre

The state is Not, if the status of the state can be changed, only when a particular resource R_i has not been selected and its status has not been changed.

Terminal states ontology:

Context State inv:

$P_s.R_i$ implies property of Resource at the start of the Activity

Context State inv:

$P_e.R_i$ implies property of Resource at the end of the Activity

Context Resource inv:

A.R implies Activity-A uses/consumes Resource-R

Context State inv:

Use

Pre: $A.R_i = true$

Post: result = $(P_s.R_i = P_e.R_i)$

The state is Use state, when a resource R_i is used by an activity-A and if none of the properties of the resource are changed when the activity is successfully terminated.

Context State inv:

Consume

Pre: $A.R_i = true$ and $P_x.R_i = true$

Post: result = $(P_s.R_i <> P_e.R_i)$ and $P_x.R_i <> true$

The state is Consume state, when any property (P_x) of a resource R_i that existed prior to the performance of the activity-A does not exist after the activity-A has been performed, i.e., if any one of the properties of the resource is changed when the activity is successfully terminated.

Context State inv:

Release

Pre: $A.R_i = true$ and $R_i = use$

Post: result = $(P_s.R_i = P_e.R_i)$

The state is Release state, when a resource R_i used by an activity-A is released and if none of the properties of the resource are changed when the activity-A is successfully terminated.

Context State inv:

Produce

Pre: $A.R_i = true$ and $P_x.R_i <> true$

Post: result = $(P_s.R_i <> P_e.R_i)$ and $P_x.R_i = true$

The state is Produce state, when any property (P_x) of a resource R_i that did not exist prior to the performance of the activity-A has been created by the activity-A, i.e., if any one of the properties of the resource is changed when the activity-A is successfully terminated.

APPENDIX B

RESOURCE ONTOLOGY

Context Resource inv:

q.(R.l.A.s.u) implies quantity q of Resource R, in terms of unit u, at location l, used by Activity A, in situation s

Context Resource inv:

$q_s.(R.A.u)$ implies quantity q of Resource R in terms of unit u at the start of Activity A

Context Resource inv:

$q_e.(R.A.u)$ implies quantity q of Resource R in terms of unit u at the end of Activity A

Context Resource inv:

$r.(R.A)$ implies role r performed by Resource R with respect to Activity A

Context Resource inv:

$R.R_P$ implies Resource R_P is a physical division of Resource R

Context Resource inv:

$R.R_F$ implies Resource R_F is a functional division of Resource R

Context Resource inv:

Use

Pre: $q_s.(R.A.u) = true$

Post: $q_s.(R.A.u) = q_e.(R.A.u)$

The use specification term entails that the resource amount will remain constant even after the resource is used, i.e., the quantity q_s of the resource-R at the start of the activity-A will be equal to the quantity q_e at the end of the activity-A.

Context Resource inv:

Production

Pre: $q_s.(R.A.u) = true$

Post: $q_s.(R.A.u) < q_e.(R.A.u)$

The production specification term entails that the resource amount will increase by a constant after the completion of the activity, i.e., the quantity q_s of the resource-R at the start of the activity-A will be less than the quantity q_e at the end of the activity-A.

Context Resource inv:

Physical_divisible

Pre: $r.(R.A) = r.((R.R_P).A)$

Post: -

A resource is physically divisible if the act of physically dividing the resource does not affect its role in the activity. Each division can be used or consumed by the same activity. A resource-R is physically divisible with respect to an activity-A if each physical division of the resource (R_P) has the same role as the whole resource-R.

Context Resource inv:

Functional_divisible

Pre: $r.(R.A) = r.((R.R_F).A)$

Post: -

A resource is functionally divisible if each division performs the same function as the whole resource. A resource R is functionally divisible with respect to an activity A if each functional division of the resource (R_F) has the same role as the whole resource R.

Context Resource inv:

Continuous

Pre: Physical_divisible = true

Post: -

A resource is continuous if it is physically divisible. Continuous resource indicates that the resource is uncountable.

Context Resource inv:

Discrete

Pre: not Continuous

Post: -

A resource is discrete if it is not continuous. Discrete resource indicates that the resource is countable.

APPENDIX C

ORGANIZATION STRUCTURE ONTOLOGY

Role ontology:

Context Role inv:

$r_1.r_2$ implies role r_2 is sub-ordinate of role r_1

Context Role inv:

if $((r_1.r_2)$ and $(r_2.at_2))$ then $(r_1.at_2)$

If role r_2 is a sub-ordinate of role r_1 and role r_2 has authority at_2, then role r_1 also has authority at_2.

Goal ontology:

Context Goal inv:

$g_1.g_2$ implies g_2 is a sub-goal of g_1

Context Goal inv:

$g.t$ implies goal g is achieved in time t

Context Goal inv:

if $(((g.g_1)$ and $(g.g_2))$ and $((g_1.t)$ and $(g_2.t)))$ then $g.t$

If both goal-g_1 and goal-g_2 are sub-goals of goal-g, and both goal-g_1 and goal-g_2 are achieved in time-t, then the goal-g is also achieved in time-t.

Agent ontology:

Context Agent inv:

oa.r implies organization agent *oa* plays a role *r*

Context Agent inv:

oa.tm implies organization agent *oa* is a member of team *tm*

Context Team inv:

tm.r implies team *tm* has a goal *g*

Context Agent inv:

if (($(oa_1.tm_1)$ and $(oa_2.tm_1)$) and ($(oa_1.r_1)$ and $(oa_2.r_1)$)) then $(tm_1.r_1)$

If both agents oa_1 and oa_2 belong to team-tm_1 and both have the same role-r_1, then the team-tm_1 has the role-r_1.

Context Agent inv:

$oa_1.oa_2$ implies agent oa_1 has authority over agent oa_2