Semantic Modeling Method for Configurable Enterprise Information Systems

QIANFU NI

B. Eng. and M. Eng. (Mfg)

This dissertation was submitted

as part of the requirements for the degree of

Doctor of Philosophy

School of Engineering Systems

Queensland University of Technology

2010

ABSTRACT

Business practices vary from one company to another and business practices often need to be changed due to changes of business environments. To satisfy different business practices, enterprise systems need to be customized. To keep up with ongoing business practice changes, enterprise systems need to be adapted. Because of rigidity and complexity, the customization and adaption of enterprise systems often takes excessive time with potential failures and budget shortfall. Moreover, enterprise systems often drag business behind because they cannot be rapidly adapted to support business practice changes. Extensive literature has addressed this issue by identifying success or failure factors, implementation approaches, and project management strategies. Those efforts were aimed at learning lessons from post implementation experiences to help future projects. This research looks into this issue from a different angle. It attempts to address this issue by delivering a systematic method for developing flexible enterprise systems which can be easily tailored for different business practices or rapidly adapted when business practices change.

First, this research examines the role of system models in the context of enterprise system development; and the relationship of system models with software programs in the contexts of computer aided software engineering (CASE), model driven architecture (MDA) and workflow management system (WfMS). Then, by applying the analogical reasoning method, this research initiates a concept of model driven enterprise systems. The novelty of model driven enterprise systems is that it extracts system models from software programs and makes system models able to stay independent of software programs. In the paradigm of model driven enterprise systems, system models act as instructors to guide and control the behavior of software programs. Software programs function by interpreting instructions in system models. This mechanism exposes the opportunity to tailor such a system by changing system models. To make this true, system models should be represented in a language which can be easily understood by human beings and can also be effectively interpreted by computers. In this research, various semantic representations are investigated to support model driven enterprise systems.

The significance of this research is 1) the transplantation of the successful structure for flexibility in modern machines and WfMS to enterprise systems; and 2) the advancement of MDA by extending the role of system models from guiding system development to controlling system behaviors. This research contributes to the area relevant to enterprise systems from three perspectives: 1) a new paradigm of enterprise systems, in which enterprise systems consist of two essential elements: system models and software programs. These two elements are loosely coupled and can exist independently; 2) semantic representations, which can effectively represent business entities, entity relationships, business logic and information processing logic in a semantic manner. Semantic representations are the key enabling techniques of model driven enterprise systems; and 3) a brand new role of system models; traditionally the role of system models is to guide developers to write system source code. This research promotes the role of system models to control the behaviors of enterprise.

Keywords:

Model Driven System, Semantic Representation, Semantic Modeling, Enterprise System Development, EAI, ERP, MDA

DECLARATION

I declare that the ideas, experimental work, analyzes, software and conclusions reported in this dissertation are entirely my own effort, except where otherwise acknowledged. I also certify that the work is original and has not been previously submitted for any degree or diploma.

None of the materials contained in this thesis have been submitted for publication prior to the start of candidature. Some of the work in the thesis has been published or submitted for publication in refereed journals or international conferences prior to the completion of this thesis.

Signed: .... **Date:** ....

ACKNOWLEDGEMENT

I would like to give my heartfelt thanks to my principal supervisor Professor Prasad Yarlagadda for his pertinent guidance, support and encouragement during my candidature. It wouldn’t have been possible to complete without his advice, encouragement and support.

I also give my thanks to my associate supervisor Dr Mahalinga Mahalinga-Iyer, who was always willing to help.

Thanks are given to my external supervisor, Professor Wen Feng Lu, who gave me much technical advice and spent much precious time reviewing this thesis and the associated technical papers.

Thanks to the School of Engineering Systems and the Faculty of Built Environment and Engineering for providing this opportunity to make this PhD a reality.

In conclusion, thanks to all the people who have encouraged and helped me in my study, but are not named here.

DEDICATION

This thesis is dedicated to my wife, Sandra, who took over my share of housekeeping when I sat in the front of the computer; Dedication is also given to my son Jingwei and daughter Yvonne who are a great source of motivation and inspiration.

PUBLICATIONS

Journal Papers

Ni, Q.F., Yarlagadda, P. and Betts, M., 2009, An Extensible Manufacturing Resource Model for Process Integration, *Asian International Journal of Science and Technology: Production and Manufacturing*, Vol. 4, No. 1, pp1-16

Ni, Q.F., Lu, W.F. and Yarlagadda, P., 2008, An extensible product structure model for product lifecycle management in the make-to-order environment, Concurrent Engineering: Research and Applications, Vol. 6, No. 4, pp243-251.

Ni**,** Q.F., Yarlagadda, K.D.V. Prasad and Lu, W.F., 2007, A Configuration-based Flexible Reporting Method for Enterprise Information Systems, *Computers In Industry*, Vol. 58, No. 7, pp416-427.

Ni**,** Q.F., Lu, W.F, Yarlagadda, K.D.V. Prasad and Ming, X.G., 2007, Business Information Modeling for Process Integration in the Mould Making Industry, *Robotics and Computer-Integrated Manufacturing*, Vol.23, No.1, pp195-207.

Ni, Q.F., Lu, W.F., Yarlagadda, P., and Ming, X.G. 2006, A collaborative engine for enterprise application integration, *Computers in Industry*, Vol. 57, No. 7, pp640-552.

Ni, Q.F., Lu, W.F. and Yarlagadda, P., 2006, A PDM-based Framework for Design to Manufacturing in Mould Making Industry - a Case Study of Business Process Integration, *Computer-Aided Design and Applications*, Vol. 3, No. 1-4, pp211-220.

Ni, Q.F., Yarlagadda, P. and Lu, W.F., 2006, Modeling of an Integrated Process Planning, *Computer-Aided Design and Applications*, Vol. 3, No. 5, pp567-576.

Conference Papers

Ni, Q.F., Yarlagadda, P., and Lu, W.F., 2006, Semantic Representations for Configurable Enterprise Systems, *Proceedings of IDETC/CIE 2006 ASME 2006 International Design Engineering Technical Conferences & Computers and Information in Engineering (IDETC/CIE) Conference*, Pennsylvania USA, September 10-13, 2006.

Ni, Q.F. and Yarlagadda, K.D.V. Prasad, 2006, A Configuration-based Method for Developing Enterprise Information Systems, *Proceedings of BEE Conference*, Queensland Australia.

Ni, Q.F., Yarlagadda, P. and Lu, W.F., 2005, Product Structure Modeling for the Made-to-Order Environment, *Proceedings of IDETC/CIE 2005 ASME 2005 International Design Engineering Technical Conferences & Computers and Information in Engineering (IDETC/CIE) Conference*, California USA, September 24-28, 2005.

Ni, Q.F., Yan, J.Q. and Ming, X.G., 2004, Configurable Resource Model for Process-oriented Applications, *Proceedings of the 11th International Conference on Concurrent Engineering (ISPE), Beijing*, China, July 2004.

CONTENTS

ABBREVIATIONS

ADL – Attribute Declaration Language

API – Application Programming Interface

ARIS – Architecture of Integrated Information System

B2B – Business to Business

BLOB –Binary Large Object

BOM – Bill of Material

CAD – Computer Aided Design

CAM – Computer Aided Manufacturing

CAPP – Computer Aided Process Planning

CCID – China Center for Information Industry Development

CNC – Computer Numerical Control

COM – Component Object Model

CORBAR – Common Object Request Broker Architecture

CASE – Computer Aided Software Engineering

CEPC – Configurable Event Driven Process Chain

CIMOSA – Open System Architecture for Computer Integrated Manufacturing

CRM – Customer Relationship Management

DAIM – Design-Analysis Integration Model

DICOM – Digital Imaging and Communication

EAI – Enterprise Application Integration

ebXML – Electronic Business using Extensible Markup Language

EC – Engineering Change

EJB – Java Enterprise Bean

EPC – Event Driven Process Chain

ERA – Enterprise Reference Architecture

ERP – Enterprise Resource Planning

FMS – Flexible Manufacturing System

GIM – GRAI integrated methodology

GUI – Graphical User Interface

HTTP – Hypertext Transfer Protocol

IDC – International Data Corporation

IDL – Interface Declaration Language

IS – Interface Declaration Language

IT – Information Technology

J2EE – Java 2 Enterprise Edition

MDA – Model Driven Architecture

MRP – Manufacturing Resource Planning

NC – Numerical Control

OAM – Open Assembly Model

OLE – Object Linking and Embedding

OMG – Object Management Group

OOP – Object Oriented Programming

PDM – Product Data Management

PERA – Purdue Enterprise Reference Architecture

PFEM – Product Family Evolution Model

PIM – Platform Independent Model

PLM – Product Lifecycle Management

PR – Purchase Request

PSM – Platform Specific Model

POI – Poor Obfuscation Implementation

PTC – Parametric Technology Corporation

RFQ – Request for Quotation

SCM – Supply Chain Management

SDLC – Software Development Lifecycle

SOA – Service Oriented Architecture

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

STEP – Standard for the Exchange of Product Model Data

UDDI – Universal Description, Discovery and Integration

UML – Universal Modeling Language

WAPI – Workflow Application Programming Interface

WfMC – Workflow Management Coalition

WfMS – Workflow Management System

WSDL – Web Services Description Language

XML – Extensible Markup Language

XSL – Extensible Stylesheet Language

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

Computer application in enterprises has gone through three major milestones. The first milestone is endorsed by a standalone manner in which computers worked. At this stage computers were mainly used as a means to assist individuals. Typical software applications were various computer-aided applications, such as computer aided design (CAD), computer aided process planning (CAPP) and computer aided manufacturing (CAM). A study uncovered that the performance improvement of individual activities does not contribute to the overall performance of an enterprise (Abeysinghe and Phalp 1997). This is evidenced through the complaints of some companies that their business performance was not improved as expected though adopted technologies functioned well. The findings of this research unveiled the main factors that led to this result, which are as follows:

- Applications were selected and implemented based on the initiatives of individual departments rather than the common goal of the enterprise;
- Information models underneath each application with little interoperability resulted in no way of electronically sharing or exchanging information (Kim, Kim and Choi 1993);
- Lack of integration resulted in broken information flows and fragmentized business processes;
- Inflexibility of software applications could not effectively support business practice changes for new business opportunities;
- Internal competition for investment in implementing new systems or upgrading existing systems led to difficulty to achieve the most optimal technology deployment (Hammer 2002);

- Too much emphasis was focused on the technologies and the human aspect was ignored (Sun 2000).

The age of information integration arrived when enterprises recognized the shortcomings of the discrete computing environment. Information integration was to build up fundamental information infrastructure, with new information technologies incorporated, to make information exchangeable between computer applications (Abdmouleh and Spandoni 2004). Information integration made information exchangeable by transforming information from proprietary formats to neutral formats. During this period, fundamental standards were developed and commonly accepted for representing information for exchange. Though information integration was recognized as a successful means to achieve information sharing between applications, its shortcomings, as highlighted below, are obvious:

- Success of information integration relied too much on the openness of individual applications and the maturity of various standards;
- Additional work was often required to import or export information in a neutral format;
- Coexistence of multiple copies of the same piece of information was, in many cases if not always, inevitable. The difficulty to synchronize information stood out;
- A common goal of an enterprise could not be clearly highlighted as information integration usually took place between different pairs of applications;
- Internal competition for technology upgrading among different departments still existed because information integration only focused on some applications (Liu, Wang and He 2004).

In the past decade, computer utilization in enterprises has rapidly evolved from information integration to process integration. Process integration positions individual performance initiatives under a process umbrella to maximize the overall performance (Hammer 2002). Table 1 presents the main differences between traditional enterprise and process enterprise. In nature, process integration connects technology, process and people (Vinther 2008). Process integration usually involves business process modeling, application integration, and culture transformation. Business process modeling captures the as-is business process and establishes a to-be business process model which removes business process bottlenecks and connects fragments in the existing business process. Application integration connects various applications to support the new business process model. Culture transformation wakes up people to an awareness of the business process. In a process integrated enterprise, people need to have the mindset that their goal is to help downstream teams to complete tasks better, rather than simply to complete their own work.

Table 1 Traditional enterprise versus process enterprise (Hammer 2002)

	Traditional Enterprise	**Process Enterprise**
Central Axis	Function	Process
Work Unit	Department	Team
Job Descriptions	Limited	Broad
Measures	Narrow	End-to-end
Focus	Boss	Customer
Compensation	Activity-based	Results-based
Manager's Role	Supervisor	Coach
Key Figure	Functional executive	Process owner
Culture	*Conflict-oriented*	*Collaborative*

One of the promises from process integration is to connect all activities of an enterprise. As a result, enterprises can work as a whole towards a common goal. Successful implementation of process integration offers enterprises opportunities:

- To optimize resource utilization;
- To improve cooperation, coordination and communication;
- To link functions with information, resources, applications and people;
- To streamline material, information and control flows throughout the entire business process (Ortiz, Lario and Ros 1999, Tang 2004).

Process integration can lead to significant benefits such as improved customer service, better scheduling, and reduced costs. As such, process integration has been widely adopted by enterprises to improve business performance in terms of productivity, flexibility and quality (Li, Wang, Wong and Lee 2004). Process oriented software systems are also commercially available in market under a variety of labels including enterprise resource planning (ERP), product data management (PDM), supply chain management (SCM), customer relationship management (CRM) and workflow management (Sun 2000).

1.1 **Risks in Implementation of Enterprise Systems**

Studies revealed that the number of companies that implemented process integration kept increasing. International Data Corporation (IDC) reported that ERP systems revenue was $21.5 billion in 2000 (Cowley 2010). Aberdeen Group concluded that spending in the business process management software sector reached $2.26 billion in 2001. According to CCID Report (2004), ERP sales in Mainland China reached US $226.9 million in 2003, and would reach US $652.8 million by 2008 (Zhang, Lee, Huang, Zhang and Huang 2005).

Process-oriented systems are very expensive. System costs, in general, range from hundreds of thousands of dollars to several million dollars (Rolland and Prakash 2000). Due to the complexity of such systems, the implementation cost can be much higher than the systems cost. A survey (Ross and Vitale 2000) concluded that implementation costs range from $2 million to $130 million after studying process integration projects in 15 companies, ranging in size from $125 million to over $25 billion (US) in sales, of which eight companies deployed SAP, three implemented Baan, another three used Oracle and one adopted PeopleSoft. The survey also stressed that cost may escalate when counting in costs of human resources for project implementation, new hardware for running the system, and integration with other types of applications. A study by Gartner Group revealed that consultants helping in the selection, configuration and implementation may cost up to three times much money as system cost (CTRC 1999).

Implementation of process-oriented systems is difficult and time-consuming with potential failures (Zhang, Lee, Huang, Zhang and Huang 2004). Study in the literature (Ross, et al. 2000) concluded that implementation time from the signing of the contract until the final “go-live” ranges from one to five years. Standish Group reported that ERP implementation projects, on average, were 178% over budget, took 2.5 times as long as intended and delivered only 30% of promised benefit (Williamson 1997). Nearly 1000 companies in China have implemented MRP, MRP II or ERP systems since 1980 to 2005. The successful implementation rate is extremely low at only 10% (Zhang, et al. 2005). ERP failures, cancellations, and cost/time overruns have also been reported in different studies (King 1997).

It is absolute that the world keeps changing. Enterprises may need to change business practices from time to time for new business objectives or new opportunities. Strategic flexibility is a critical competency that enterprises have to have in today's dynamic global environment. Business operation tightly relies on the support of enterprise systems. Changes to business practices often result in modifications to enterprise systems.

However, throughout process integration, computers in enterprises evolve from standalone facilities to complicated interconnected network systems. The goal of software applications shifts from assisting individuals to connecting various functional units. The enterprise itself is transformed from relatively independent departments to an interdependent environment. Since process integration results in the interconnection of most, if not all, applications, a small change to one application in a process integrated enterprise can lead to a chain of changes to many other upstream or downstream applications. Therefore, such an enterprise has to have various levels of project teams and support teams for maintaining and upgrading applications based on business needs. From the long-term viewpoint, the ongoing maintenance cost in a process integrated enterprise is really much higher than the system and implementation costs.

1.2 Research Gap Identification

Risks in the implementation of enterprise systems have drawn the attention of researchers (Botta-Genoulaz, Millet and Grabot 2005). Many research efforts have been founded on identifying success or failure factors (Bradford and Florin 2003, Sarker and Lee 2003, Umble, Haft and Umble 2003a, Calisir 2004, Zhang, et al. 2005), implementation approaches (Umble, Haft and Umble 2003b, Bendoly and Kaefer 2004), and project management strategies (Sarkis and Sundarraj 2003, Yen and Sheu 2004, Gebauer and Lee 2008). Those efforts are aimed at learning lessons from post implementation experiences and providing knowledge to help future implementation projects.

At present, customization is still the primary method of individualizing enterprise systems for different enterprises. Customization involves massive effort to redesign functions and change software codes in order to deliver tailored functions. Compromises are often necessary due to the limitation of the base system and the budget. In addition, current enterprise systems lack flexibility so they cannot be quickly changed to keep up with business changes. This necessitates the involvement of different teams, software vendors and third parties in upgrading enterprise systems and is one of the main factors that results in high maintenance cost. However, few research reports can be found on methods for developing enterprise systems with the better flexibility to support different enterprises and ongoing business changes.

System models represent business practices. Business practice differences imply model differences and business practice changes require model changes. Currently system models are hard coded into software programs and cannot exist independent of software programs. Changing software code is the only way to incorporate new models into software programs when business practices change.

This research attempts to fill the gap by initiating a concept of model driven enterprise systems. The novelty of model driven enterprise systems is that they decouple system models from software programs. System models can physically exist outside of software programs. In model driven enterprise systems, system models act as instructors to guide and control the behavior of software programs. Software programs function by interpreting instructions contained in system models. This offers the opportunity to control the behavior of software programs by changing system models. Such an enterprise system provides high flexibility and can be individualized with little modifications to software source code. Therefore, model driven enterprise systems can be rapidly configured and reconfigured to satisfy different requirements or ongoing business changes.

1.3 Research Justification

Flexibility is referred to as the ability to change or the rapid adaption to future uncertainty at minimal cost and effort with little disturbance on other performance variables (Slack 1989, Upton 1994, Volberda 1999). It has been becoming critical for enterprises to win business opportunities (Ni 2007). Enterprise systems can effectively support business operations only when they are well aligned with business requirements (Dreiling, Rosemann, Aalst, Sadiq and Khan 2006). The flexibility of enterprise systems can have important consequences for operational efficiency and long-term effectiveness, yet is often not considered explicitly as a decision factor during system design and implementation (Gebauer, et al. 2008).

Inarguably product market lifecycle is becoming shorter and customer demands are frequently changing. Enterprises are forced to optimize their business practices in order to tackle these challenges. A wide consensus has been reached, that being able to respond to market and customers at lower cost and high quality is not enough anymore. In the modern business environment, it is critical for enterprises to have the flexibility to provide new products and continuously improved customer services. Higher business flexibility can be achieved only when the enterprise system in use can support changes quickly. Though enterprise system vendors have taken flexibility as a high priority and endeavored to develop generic architecture, enterprise systems are still subject to criticism with respect to rigidity (Ni 2007). In reality, enterprise system implementation and maintenance still involves massive customization efforts to make functions satisfy specific needs.

Enterprise systems need to be re-aligned with business requirements to catch up with business changes. How easily and quickly an enterprise system can be aligned and re-aligned with business requirements, has become a key indicator of business agility. Flexibility has been a requirement of enterprises for decades but the concept still remains understood (Ni 2007). The majority of research in the past decades has focused on strategic flexibility in the context of manufacturing systems.

Enterprises still lack fundamental theories and systematic approaches to achieve and measure flexibility, particularly in the context of software development and selection.

Currently, many enterprise systems are developed, based on so-called ‘best practice’. In theory, it sounds very attractive to adopt the best practice. However, in reality, few companies want to give up their existing practices for the best practice, especially if a company has already achieved competitive advantages in enacting a business process (Scott and Vessey 2000). Therefore, individualization is still a popular option in the adoption of enterprise systems.

Table 2 presents methods currently being practiced to tailor or adapt enterprise systems (Ni 2007). A conclusion can be drawn from the table that customization is still a dominant approach in specializing or adapting an enterprise system. Customization is to build, fit, or alter according to individual specifications by changing system source code. This incurs excessive effort and implies a long implementation lifecycle. Some degree of enterprise system customization is possible, however the complexity of enterprise systems makes major modifications impractical (Davenport 1998). Therefore, enterprises often have to compromise in the adoption of enterprise systems. Therefore, of three challenges to enterprise system vendors, delivering flexibility takes the first priority (Goyal 2006).

Table 2 Methods for adapting enterprise systems (Ni, 2007)

Method	Participants			
	Adopter	Vendor	3rd Party Software Vendor	3rd Party Support
Reconfiguration	✓			✓
User Interfacing Tuning	✓			✓
Extended Reporting	✓			✓
Programming	✓			✓
Patch Upgrades	✓	✓		✓
Version Upgrades	✓	✓		✓
Acquire New Module or License		✓		✓
Acquire 3rd Party Software			✓	✓

A critical success factor in enterprise system implementation is to avoid system source code changes wherever possible by using predefined change options (Holland and Light 1990). Through the analysis of interview data and literature, Ni (Ni 2007) concluded that configuration is on the top of the list of choices in the adaptation of enterprise systems if the function can satisfactorily meet new business needs. Compared with other methods, configuration is a very cost effective solution and can be completed in a short time. Moreover, it is less demanding for competent IT people and has minimal implication on the future system maintenance.

Currently, configuration is achieved by setting various parameters. Several thousands of parameters may still be insufficient to satisfy flexibility needs because of the complexity of enterprise systems, (Dreiling, et al. 2006). Furthermore, parameter based configuration has little intuitive conceptual support. Thus, setting enterprise system parameters is a process which is error-prone and resource-intensive. The lack of object-oriented and process-centric intuitiveness makes parameter based configuration extremely difficult.

The flexibility of enterprise systems is truly realized only when stability and achievability are guaranteed deterministically. With respect to the flexibility of enterprise systems, extensive research needs to be done on fundamental theories and deterministic methodologies are to be developed. This research argues that configuration remains a better approach for the flexibility of enterprise systems. However, an easy way to realize configuration is critical to the success of this approach.

System models are the projection of business practices. Business practice differences mean the difference in system models and business practice changes, require changes to system models. It is obvious that an enterprise system can be much more flexible when the following conditions are satisfied:

- System models can physically stay outside of software programs;

- Software programs can function automatically according to updated models; and
- System models can be changed independent of software programs.

In the context of this research, an enterprise system that meets the above conditions is referred as a model driven enterprise system. This research insists that model-driven configuration is one of the best approaches in the individualization and adaption of enterprise systems. System models are the blueprint of an enterprise system with the complete context of information entities, entity relationships and functional deployment. This provides comprehensive intuitiveness to support configuration. Hence, compared to other approaches, model-driven flexibility is under control and predictable with intuitiveness. The main focus of this research is on developing a conceptual architecture of model driven enterprise systems and semantic representations of system models, which are key techniques of model driven enterprise systems.

1.4 **Research Approach**

This research is exploratory in nature. The purpose of exploratory research is to investigate little understood phenomena and identify or discover important variables to generate hypotheses for further research (Marshall and Rossman 1989). The analogical reasoning approach is adopted. This approach is appropriate as the study on methods for developing flexible enterprise systems can only be measured in a qualitative manner. Analogical reasoning is a method of processing information that compares the similarities between new and understood concepts; and then uses those similarities to gain understanding of the new concept. It is a form of inductive reasoning because it strives to provide understanding of what is likely to be true, rather than deductively proving something as fact (Boelcke 2003). The inductive process of exploration offers a rigorous approach to assist understanding complex information system (IS) project implementations (Nasirin and Birks 2002). It enables the achievement of dual objectives of rigor and relevance (Melia 1996, Fernandez, Lehmann and Underwood 2002).

By using the analogical reasoning approach, this research first investigates the knowledge of flexibility accumulated in the area of modern machines and workflow management. Then, a hypothesis is derived which proposes that an enterprise system developed in the analogical method should have similar high flexibility.

1.5 Contributions

This research is the first effort to explore a deterministic method for developing flexible enterprise systems through configuration and reconfiguration. It contributes to the area of software development from the following three aspects.

1) The first contribution of this research is the concept of model driven enterprise systems. Currently, business requirements, design decisions and developers' thinking are hard coded into enterprise systems throughout the development process. In such a way, enterprise system models merge into software programs and cannot stand independent of software programs. After a system is developed, system models become intangible. This eliminates the possibility of adjusting enterprise systems by changing system models. Changes to system models need to be implemented by revising system source code. The concept of model driven enterprise systems provides an effective mechanism to separate system models from software programs and makes system models able to stay outside of software programs. The separation of system models from software programs exposes an opportunity to mediate the behavior of enterprise systems through modifying system models.
2) The second contribution is semantic representations. Traditionally, system models exist dependent of software programs. They are reflected in system source code. Model driven enterprise systems require system models to be extracted from software programs. Two key questions are raised: 1) what can be extracted from software programs as system models; and 2) how these system models are represented. This research identifies the system models to be extracted by developing an abstraction model of enterprise

systems. Then, various semantic representations are developed. These representations are an innovative technique used to represent business entities, entity relationships and processing logics. Semantic representations enable system models to exist outside of, and be loosely coupled with, software programs. They can be easily constructed by human beings. At the same time, they can also be effectively interpreted by computers.

3) The contribution is the promotion of the role of system models from guiding writing system source code to controlling the behavior of enterprise. The software development lifecycle (SDLC) provides a philosophy to manage the process of enterprise system development. In SDLC, business requirement collection, system design and system coding are major steps in ensuring that an enterprise system is developed in line with business requirements. Traditionally, software developers write system source code by understanding business requirements and design decisions. After being developed, an enterprise system works by following the way that the developers defined. In other words, developers' thinking is implanted into enterprise systems. Developers' thinking is the understanding of business requirements and design decisions. Business requirements and design decisions are usually represented as various system models. Consequently, the major role system models play is to guide developers to write system source code. MDA is a model driven framework for software development. MDA promotes the role of system models for generating system source code. This results in the synchronization of system source code with system models. The concept of model driven enterprise systems extends MDA and further promotes the role of system models for driving the behavior of enterprise systems. In nature, the concept of model driven enterprise systems is to achieve the synchronization of the behavior of enterprise systems with system models at runtime.

1.6 Structure of the Thesis

This dissertation is to explore a systematic method for developing flexible enterprise systems. This research is qualitative in nature. By applying analogical reasoning, it develops the method through studying known phenomena in relevant areas. Firstly, the dissertation investigates methods for flexibility in manufacturing and workflow management; and the role of system models in the development of enterprise systems. Throughout this process, the dissertation identifies the root cause of the inflexibility of current enterprise systems. Then, a concept of model driven enterprise systems is initiated to move away the barrier to flexibility in current enterprise systems. The concept of model driven enterprise systems requires system models to be separated from enterprise systems and represented in a semantic manner. Various semantic representations are developed to support the development of model driven enterprise systems. After that, three case studies are identified and conducted. A proof-of-concept system is also developed to demonstrate the development of model driven enterprise systems and effectiveness of the semantic model representations. Finally, conclusions and future work are presented. This chapter provides a comprehensive overview of computer utilization in enterprises. It presents flexibility needs, research gaps and research justification. It also outlines the research goal, research approach, and key contributions. The following chapters are organized as follows.

Chapter 2 Literature Review provides detailed reviews of existing research on flexibility in the context relevant to enterprise systems. Prior to the detailed review, a short review of research in manufacturing flexibility is also presented because research on flexibility originated from manufacturing;

Chapter 3 Concepts of Model Driven Enterprise Systems investigates technologies for flexibility in the areas of modern machines and workflow management. The role of system models in the development of enterprise systems is reviewed. Then, the concept of model driven enterprise system is developed. System models to be extracted are identified, and a reference architecture for model driven enterprise is discussed.

Chapter 4 Semantic Model Representations develops semantic representations for system models identified in Chapter 3. Semantic representations are a key enabler of flexible enterprise systems.

Chapter 5 Industrial Case Identification identifies industrial cases for studying the application of semantic representations in business object models by developing a comprehensive business process model based on the practice of manufacturing industry. Resource management, product structure management and reporting are selected as industrial cases.

Chapter 6, Chapter 7 and Chapter 8 are case studies which apply semantic representations to the resource model, product structure model and reporting model.

Chapter 9 Prototype develops a proof-of-concept system to further verify the effectiveness of the concept of model-driven enterprise systems and semantic representations.

Chapter 10 Conclusions and Future Research summarizes the dissertation and conclude contributions. The future research is also presented.

Chapter 2 Literature Review

2.1 Introduction

An enterprise system needs to be deployed to different enterprises and it should be able to be adapted easily when business practices change. Thus, enterprise systems have to be flexible. The easiest way to tailor and adapt enterprise systems is a configuration which involves little change to system source code. An enterprise system that offers flexibility by configuration is defined as a configurable enterprise system. Flexibility is a requirement and configuration is one method to satisfy the requirement of flexibility. Flexibility is a multidimensional concept and it has to be discussed in a particular context (Evans 1991, Ni 2007, Maksimovic and Lalic 2008, Stevenson and Spring 2009). This chapter provides detailed reviews on flexibility in the context of enterprise systems. Before proceeding to the detailed review, a short review of research in manufacturing flexibility is presented, since research on flexibility originated from manufacturing.

2.2 Research on Manufacturing Flexibility

The concept of flexibility is not a recent phenomenon. Research on flexibility in business operation has been conducted for over sixty years (Golden and Powell 2000, Stevenson, et al. 2009) and carried out from various domain areas, most notably manufacturing (Gupta and Goyal 1989b, Vokurka and O'Leary-Kelly 2000). The need for flexibility is grounded in the need to cope with uncertainty in the manufacturing environment (Schmenner and Tatikonda 2005). With the advance of new software (e.g. CAD, CAM), improved manufacturing facilities (e.g. CNC, robots, FMS), and expanded manufacturing information systems (e.g. MRP, MRP II, ERP), enterprises need to understand the potential flexibility of process and information technology deployed in their manufacturing systems to gain maximal benefits (Slack 1989). In the 1980s, flexibility in manufacturing became an outstanding issue. It was recognized that manufacturing flexibility needs to be conducted on a more scientific basis (Schmenner, et al. 2005). Gerwin is notable for his contribution to initiating research on flexibility in manufacturing (Ni 2007). He defined flexibility as the ability to respond effectively to changing circumstances; and pointed out pointed out that flexibility associates with both operational uncertainties and process design. He started measuring manufacturing flexibility using a range of possibilities that manufacturing processes can handle - time and cost, achievability and effectiveness. Continuing Gerwin's initiative, research in manufacturing has been conducted from different angles, as shown in Table 3. Different kinds of uncertainty drive the needs for different kinds of flexibility and research on flexibility has moved forward beyond manufacturing.

Table 3 Summary of research on manufacturing flexibility (Ni, 2007)

	Focus	**Literature**
General aspects	Definition & classification and interrelationship between different types of flexibility	(Browne, Dubois, Rathmill, Sethi and Stecke 1984, Slack 1987, Gupta, et al. 1989b, Slack 1989, Sethi and Sethi 1990, Gerwin 1993, Upton 1994, Cheng, Simmons and Ritchie 1997, Parker and Wirth 1999, Narain, Yadav, Sarkis and Cordeiro 2000)
	Measurement of flexibility	(Gupta, et al. 1989b, Slack 1989, Sethi, et al. 1990, Gerwin 1993, Parker, et al. 1999, D'Souza and Williams 2000)
	Factors and methods of flexibility	(De Meyer, Nakane, Miller and Ferdows 1989, Hill and Chambers 1991, Narain, et al. 2000)
Specific aspects	Manufacturing processes	(Gerwin 1987, Upton 1997)
	Human resources management	(Gupta 1989, Denton 1994)
	Business process reengineering	(Zhang and Cao 2002)
	Product design and development processes	(Sanchez and Mahoney 1996)
	Administrative aspects of flexibility	(Kathuria 1998)
	Manufacturing resources management	(Correa and Slack 1996)
	Machine adaptability	(Mandelbaum and Brill 1989, Brill and M 1990)

2.3 Need for Flexibility in Enterprise System

Change is inevitable. Change can bring risks to those unprepared, or open up new avenues of business to those best able to take advantage of the opportunity it brings (Brown 2000). The effective way of managing changes, which leads to successful business performance, calls for flexibility (Ni 2007). Flexibility enables enterprises to absorb variation and uncertainty of business process (Applegate, McFarlan and McKenney 1999, Gorod, Gandhi, Sauser and Boardman 2008). Business operation flexibility depends on:

- Strategies, such as inventory level, approval process, and material order frequency and quantity;
- Resources, such as availability, capability, and capacity;
- Technologies, such as stability, scalability, portability, maintainability and reusability;
- Production line, such as the ability to adapt to short-runs, variable demands and the variety of products.

Over the past decade, process integration has been broadly adopted as an effective solution for better flexibility to tackle the changes of business environments (Agerfalk , Smith and Fingar 2003). The goal of process integration is to make an enterprise behave as a whole towards a common objective. Process integration brings up two major changes to enterprise: organization of business units and technology deployment. Organizational restructuring makes enterprises ready for business environment changes and sophisticated technology deployment transforms this readiness to the real flexibility. Process integration transforms an enterprise from relatively independent departments to an interconnected environment. Meanwhile, it makes business operations rely largely on the support of enterprise systems. Business operation flexibility drives the demands of flexible enterprise systems. Insufficient flexibility of an enterprise system can limit the success of an enterprise because the system cannot support certain circumstances where manual handling is necessary (Gebauer and Schober 2006). In addition, inflexibility often leads to long implementation time, high customization cost, and short system lifetime (Silver 1991). This also generates indirect impacts on the flexibility of business operation. Therefore, flexible enterprise systems have become a strong business requirement, and this is drawing the attention of academia and software vendors.

2.4 **Strategic Research on Flexibility of Enterprise System**

There are risks associated with the flexibility of enterprise systems. The formulation and implementation of efficient strategies for flexibility have become important aspects of risk management (Schober and Gebauer 2008). Excessive flexibility might overwhelm enterprises (Ozer 2002). Flexibility can limit the success of an enterprise system by reducing usability (Silver 1991, Chen, Sun and Jih 2009) and increasing complexity (Anonymous 1999). Flexibility requires stability to avoid chaos (De Leeuw and Volberda 1996, Volberda 1996, 1999). Without the concern of stability, flexibility may cause overreaction and resource wastage (Volberda 1996). At the strategic level, research on the flexibility of enterprise systems mainly puts focus on business drives for flexibility, cost and risk associated with flexibility, and measurement of flexibility.

Brown (Brown 2000) identified three major business environment changes that can generate significant impacts on enterprise systems: 1) government policies and practices; 2) organization acquisition (including mergers and takeovers); and 3) major political and economic events. Fitzgerald (Fitzgerald 1990) also pointed out three types of changes which require the enhancement of enterprise systems. They are environmental, organizational and technical. Environmental changes include government legislation, industrial relations, and external agencies. Organizational changes include influences from strategy, policy, organizational structure, and procedures etc. These changes can generate major impacts on business operation. Enterprises have to concern themselves with how their enterprise systems can be changed to adapt to these changes.

Flexibility is not free and it comes at a cost (Carlsson 1989, Das and Elango 1995, Schober, et al. 2008). Gebauer and Schober (Gebauer, et al. 2006) argued that the economics of enterprise system flexibility have received comparatively little attention. The guidelines regarding the management of the economics of enterprise system flexibility are based on factors such as short-term political considerations, risk aversion, tight budgets, and “me-too” desires. Such a non-systematic and unstructured analysis approach often leads to suboptimal results (Robinson and Pawlowski 1999). The characteristics of a business process are represented by uncertainty, variability and time-criticality. Based on this fact, Gebauer and Schober (Schober, et al. 2008) developed a deterministic model for evaluating the correlation between the flexibility of enterprise systems and cost efficiency. As shown in Figure 1, the model adopts three variables: uncertainty, variability and time-criticality. Uncertainty is defined as the degree to which a process task is known to the developers of an enterprise system at the time of system initialization. Variability refers to the degree to which process activities concentrate on certain process tasks. Time-criticality measures the share of time-critical process activities. This model is intended as a quantitative measurement for enterprises to preview the cost efficiency of enterprise system flexibility.

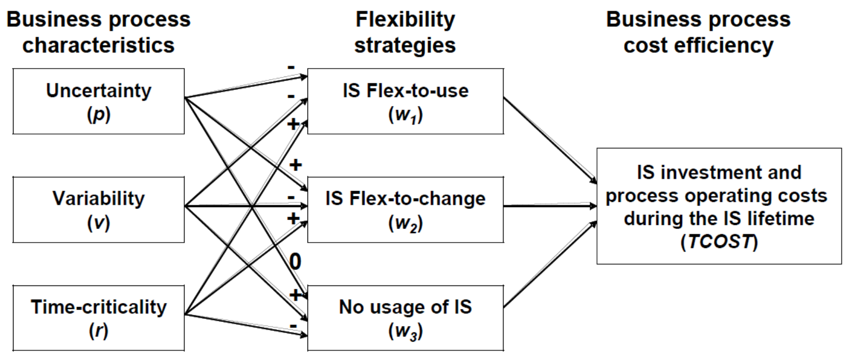

Business process characteristics
Flexibility strategies
Business process cost efficiency
Uncertainty (p)
Variability (v)
Time-criticality (r)
IS Flex-to-use (w_1)
IS Flex-to-change (w_2)
No usage of IS (w_3)
IS investment and process operating costs during the IS lifetime (*TCOST*)
-
-
+
+
-
+
0
+
-

Figure 1 Flexibility and cost efficiency (Schober, et al. 2008)

Ni (Ni 2007) also developed qualitative flexibility metrics through the study of the post implementation experience of the three partners. His metrics measure flexibility from five perspectives, as illustrated in Table 4. Versatility is to measure the capacity of flexibility, which means the ability to adapt to a range of states (Slack 1989) and accommodate a set of variety (CBDi Forum 2001). Versatility can be understood as a collection of options for decision making (Gupta and Buzacott 1989a). In this sense, versatility represents the band breadth of flexibility. Effectiveness reflects the quality of flexibility. Responsiveness measures the speed of transition. Das and Elango (Das, et al. 1995) define the responsiveness as the nimbleness and swiftness of an organization to explore external opportunities. Thriftiness looks at flexibility from the cost perspective. It manifests the ability of a system to minimize the economic values for transition. Resilience determines the ability of a system to moderate disturbance on the business performance of the organization caused by transition.

Table 4 A metrics of enterprise system flexibility (Ni 2007)

Context	Metrics	Object of Measurement
The ability of Transition	Versatility	System
	Effectiveness	System
The Cost of Transition	Responsiveness	Time
	Thriftiness	Financial cost
The Impact of Transition	Resilience	Business performance

Strategic research on flexibility relevant to enterprise systems is really literature intensive. It provides enterprises with the awareness of flexibility by addressing: 1) the effects of enterprise systems on organizational flexibility; 2) the competitive advantage to business operation (Palanisamy and Sushil 2003); and 3) the typical contradiction of enterprise systems flexibility (Allen and Boynton 1991, Robey and Boudreau 1999). Garavelli provided a comprehensive summary of strategic research on the flexibility of enterprise systems, as illustrated in Table 5.

Table 5 Strategic research on the flexibility of enterprise systems (Garavelli 2003)

Aspects	Remarks	Literature
Functional	Flexibility in operations, marketing and logistics	(Kim 1991, Lynch and Cross 1991)
Hierarchical	Flexibility at shop, plant or company levels	(Slack 1987, Gupta 1993, Koste and Malhotra 1999)
Measurement	Global flexibility measures vs. context specific measures	(Chung and Chen 1990, Gupta and Somers 1992, de Groote 1994, Sarker, Krishnamurthy and Kuthethur 1994)
Strategic	Strategic relevance of flexibility	(Nakane and Hall 1991, Chambers 1992, Gerwin 1993)
Time horizon	Long-term vs. short-term flexibility	(Zelenovich 1982)

Operational	Flexibility of product, variety and volume	(Browne, et al. 1984, Sethi, et al. 1990, Hyun and Ahn 1992, Gerwin 1993, D'Souza, et al. 2000, Vokurka, et al. 2000)

2.5 Technical Research on Flexibility of Enterprise System

With the awareness of flexibility, enterprises started fundamentally rethinking the way the work is done and proactively redesigning their business processes. Technical research on the flexibility of enterprise systems attempts to develop methods and tools to support the rethink and redesign of business process. The most fundamental technical research is on enterprise reference architecture (ERA) which is aimed at the support of business process re-engineering and advanced technology deployment.

Designing or selecting an enterprise system requires the study of an enterprise at an abstraction level. This ensures that an enterprise system designed or selected satisfies the need of the enterprise. An effective means is needed to represent enterprises for such studies. Research on ERA was initiated with this initiative. ERA concerns the current state (or as-is model) of an enterprise as well as the desired state (or to-be model) and the migration path from the as-is state to the to-be state (David Chen, Doumeingts and Vernadat 2008). ERA provides an appropriate architecture representation formalism to support the characterization of features and properties of enterprises (David Chen, et al. 2008).

One well-known research on enterprise reference architecture is the ESPRIT program carried out by the AMICE consortium. From 1985 to 1995, more than 21 companies and research units from seven European countries directly contributed to this program, plus additional partners in validation or sister projects (e.g. VOICE, CODE, CIMPRESS) (Ortiz, et al. 1999, Chalmeta, Campos and Grangel 2001). The main deliverable of this program is an enterprise reference architecture known as Open System Architecture for Computer Integrated Manufacturing (CIMOSA). CIMOSA provides a complete framework for analyzing, modeling and designing enterprises. It enables operational integration to be achieved based on four abstract views and three modeling levels. The four views are function, information, resource and organization views; and three modeling levels are requirements definition, design specification and implementation description. A detailed review of CIMOSA can be found in the literature (Bernus and Nemes 1997, Hanneghan, Merabti and Colquhoun 2000, Shin and Leem 2002).

Another famous research project conducted at Purdue University also led to a generic enterprise reference architecture, named as Purdue Enterprise Reference Architecture (PERA). PERA represents manufacturing enterprises using two functional streams, which are the information stream and the manufacturing/customer service stream. These two streams are mainly employed for describing tasks and functions of the enterprise. They are further rearranged into three implementation sets of functions: 1) human activities related to information and manufacturing/customer service; 2) information stream activities not carried out by humans; and 3) manufacturing/customer service activities not carried out by humans (Li and Williams 1997, Shen, Wall, Zaremba, Chen and Browne 2004).

In addition to CIMOSA and PERA, ARIS, GIM, GRAI and Zachman are well known enterprise reference architectures which are widely recognized (Noran 2003). Comparing these architectures, CIMOSA and ARIS present a strong similarity and are both process oriented approaches aimed at integrating functions by modeling and monitoring the action flow. GIM is based on the GRAI decision model where integration is seen as the coherence between global and local decision objectives. PERA and Zachman architectures do not provide any new modeling formalism but define complex architecture frameworks. All these architectures are heterogeneous and complementary rather than contradictory (David Chen, et al. 2008).

On the top of the fundamental research, extensive studies can be found based on the results of the fundamental research. In the middle of 1990s, the IFAC–IFIP Task Force undertook preliminary work to identify and analyze redundancies and complementarities for possible harmonization of different architectures. The IRIS group from University Jaume I of Castellón in Spain also reported an enterprise reference architecture for integrated development, named ARDIN which is the

Spanish acronym of enterprise reference architecture. Complimenting other architectures, this architecture is capable of supporting single-enterprise integration and virtual enterprise. It looks at enterprises from five aspects including enterprise development methodology, the enterprise integrated model, enterprise structures, supporting tools and efficient change management. The two major advantages of ARDIN are its strongly practical application to enterprises and emphasis on continuous change management (Chalmeta and Grangel 2003).

Dreiling et al extended the event driven process chain (EPC) by adding a set of new notations to ARIS. The extension resulted in a configurable process model language, named configurable event driven process chain (CEPC), which can be effectively used to represent process alternatives (Dreiling, et al. 2006). Further research has been conducted to identify generic configuration patterns and fundamental theories for verifying configurable process models and exploring the flexibility potential of process models. Similar research in this area also can be found on model formalization, simulation and verification (van der Aalst 1999).

Compared to strategic research on the flexibility of enterprise systems, technical research provides enterprises with systematic methods and tactical tools for representing, analyzing and designing business processes. Current enterprise systems vary from one with little room for subsequent change, to a system with many options for future change (Rumbaugh, Blaha, Premerlani, Eddi and Lorensen 1991). It is challenging to select a suitable enterprise system and to fully use the potential of a selected enterprise system. Enterprise reference architecture fills this gap and creates values to enterprises from the following aspects:

1) Reference architecture provides a clear blueprint of transition from the as-is state to the to-be state This helps enterprises select a suitable enterprise system with sufficient flexibility to support the transition;
2) Reference architecture describes the basic arrangement and connectivity of parts of an enterprise system (David Chen, et al. 2008). It is the foundation of enterprise systems' engineering and a means to assist stakeholders to manage system engineering and changes;

3) Reference architecture assists the exploration of flexibility potentials of deployed enterprise systems.

2.6 Innovations for Flexibility of Enterprise Systems

Enterprise systems need to offer a lot of functionality in order to cope with a large number of business requirements. Functionality needs to be aligned with business operation to create values for enterprises. This alignment is not ongoing after it is achieved because business operations keep changing (Dreiling, et al. 2006, Holschke, Rake, Offermann and Bub 2010). Enterprise systems must change to provide appropriate functionality when changes occur in business environments (Brown 2000). For many years, the commonly used approach in achieving this alignment is to customize functions or modules. This approach is not permanent because it only provides temporary relief to organizations. Customization is time consuming and resource intensive because it often involves different teams, vendors and other third (Martinho 2010). Driven by industrial needs, two recent interrelated innovative technologies have been developed to enable enterprise systems to be changed quickly. These are service oriented architecture (SOA) (Bieberstein, Bose, Fiammante, Jones and Shah 2005, Walsh 2010) and Web services (Whiting 2003).

Platform independency was one of the big barriers in the interoperability of enterprise systems (Wada, Suzuki and Oba 2008). For example, a program written in Java, no matter how well it is designed and how efficiently it fulfills its purpose, it is only useful on the Java platform (Footen and Faust 2008). Tight coupling between applications and platforms results in the inability of different applications to communicate with each other, and in turn, the rigidity of business operation (Margaria and Steffen 2009). SOA achieves interoperability by employing two architectural constraints: 1) the functionalities should be exposed as a set of standardized interfaces. These interfaces should be available globally to all service consumers and other service providers, regardless of their platforms; and 2) contract schemata delivered to interfaces should be descriptive and extensible. SOA is a framework that provides common and reusable business functionalities as a set of services in a standardized way. It provides the flexibility for enterprises to enhance interoperability between different applications.

SOA is the result of rethinking computing architecture (Footen, et al. 2008). The rethinking introduces the concept of encapsulation by learning from the object oriented programming (OOP) philosophy. Encapsulation in OOP means that an object exposes interfaces for other software objects to use, but implementation details are hidden (McCarty and Cassady-Dorin 1999). In other words, interfaces are the abstraction of functions an object provides to other objects. Other objects can use these functions without the need to know how those functions are implemented internally. This mechanism provides high independency among objects. SOA introduces application level encapsulation to software architecture. Each application can expose a set of services for other applications to use. Similar to OOP, a service is the abstraction of reusable units, which exposes business functions in a standardized way for other applications.

SOA is a design methodology which enables the combination of loosely coupled applications into a unified, service-based infrastructure. In order to enable service exposure, an SOA technology must represent service interfaces in an appropriate way for two services to communicate. The first pilot to really try to solving this problem was Common Object Model (COM) by Microsoft (Footen, et al. 2008). COM introduces interface definition language (IDL) for COM components to define operations to be exposed. In addition to COM, the common object request broker architecture (CORBA) is another alternative to support SOA. CORBA was developed by the Object Management Group (OMG), a consortium made up of a number of technology companies and dedicated to the development of object-oriented concepts to the enterprise. CORBA is more flexible than COM and took off in popularity beyond COM. CORBAR enables an enterprise application to be developed in any methodology desired; then the application can be wrapped into a standard interface using an IDL as a service. Therefore, CORBA is considered as the first implementation of enterprise SOA.

Nowadays, Web services have replaced COM and CORBA as a preferred SOA technology. It has resulted in a multi-billion software industry sector (Footen, et al. 2008). Web services have an extensible markup language (XML) based communication protocol for exchanging messages between loosely coupled applications. The Web services framework is divided into three areas: 1) communication protocols; 2) service descriptions; and 3) service discovery. The following specifications are currently very stable in each area (Wang, Huang, Qu and Xie 2004): 1) simple object access protocol (SOAP) that enables communications among Web services; 2) Web Services Description Language (WSDL) that provides a formal and computer-readable description of Web services; and 3) universal description, discovery and integration (UDDI) for service directory that is a registry of Web services descriptions. Web services becomes popular because it leverages a set of common industrial standards including HTTP, XML, XML Schema, SOAP and UUDI. Research in Web services is literature intensive. Active research in this area includes standardization, semantic web, grid services and web service security (Wang, et al. 2004).

2.7 **Methods for Development of Flexible Enterprise Systems**

2.7.1 Model Driven Architecture (MDA)

MDA, initiated by OMG, is a prominent effort in the method of enterprise system development (Xiao and Greer 2009). It brings up a new approach for understanding and developing complex systems(Lings 2009). It promotes portability across enterprise system platforms which can be in use now or may be invented in the future (Aagedal, Bézivin and Linington 2005, Touzi, Benaben, Pingaud and Lorré 2009). This is achieved by introducing tools of model transformation. MDA consists of three main elements: 1) PIMs (platform independent models); 2) PSMs (platform specific models); and 3) model transformation. PIMs are technology-independent models, such as models in unified modeling language (UML). A PIM depicts business processes, entities and objects and their interaction rules in a business domain (Heckel and Lohmann 2003, Kaim, Studer and Muller 2003, Gracanin, Singh, Bohner and Hinchey 2004). It is the requirement representation in a business language without technological details. PSMs are technology-dependent models based on a real technical platform, such as CORBA, J2EE or .NET. A PSM contains platform specific information, such as EJB or CORBA stubs. It is a further description of a related PIM with more technical details. Model transformation is a means to convert PIMs to PSMs. The major initiatives of this concept are: 1) enable domain experts to formulate business requirements in a familiar and platform-independent format; and 2) automate the generation of software artifacts based on PIMs.

A major advantage MDA offers, is that a set of PSMs at different abstraction levels can be implemented based on different platforms with the integrity of the entire application preserved. For example, an internal banking PIM may be realized using an EJB while a business to business (B2B) PIM could be realized by using SOAP. However, these two PSMs can interoperate with each other as defined in the PIMs. For a certain reason, if the B2B PSM needs to be implemented using CORBA, it would have no impact to any other models.

The research in MDA mainly puts focus on discussing some key concepts, such as model mapping and system generation. Kent (Kent 2002) explored the application of MDA to software system engineering to achieve model driven engineering (MDE). He discussed various problems that would occur when applying MDA to software system engineering. Edwards (Edwards, Deng, Schmidt, Gokhale and Natarajan 2004) recognized that middleware platforms, such as CORBA and Component Object Model (COM), lack a simple and intuitive way to support service configurations and deployments. He developed generative model-driven techniques and tools to automate tasks related to service configuration and deployment. These tasks are often associated with the integration of publish/subscribe services into a component-based system. They evaluated the tools based on a real-time system with over 50 components. Bauer (Bauer, Müller and Roser 2004) developed a methodical approach for integrating cross-enterprise business processes based on MDA.

From the viewpoint of flexibility, MDA synchronizes business requirements with enterprise systems through automated system generation. When business requirements change, corresponding PIMs are revised. Then, system generation tools can be run again to regenerate system source code. MDA maintains the consistency between business requirements and system source code. This can remarkably shorten the cycle from system analysis to system source code change.

2.7.2 Workflow Management

2.7.2.1 Overview of Workflow Management

At present, enterprise system configuration is a process of switching on or off functions (Dreiling, et al. 2006). Much research in this area is still conducted at strategic and tactical levels. Research on deterministic methods for developing enterprise systems with flexibility is rarely found. Based on this literature review, workflow is a technology closely related to the development of flexible enterprise systems.

According to workflow management coalition (WfMC), workflow is defined as the computerized facilitation or automation of a business process, in whole or part. A workflow management system (WfMS) is a system that completely defines, manages and executes workflow through the execution of software driven by a computer representation of the workflow logic (WFMC, 1995). WfMC has developed a standard framework as a platform for describing the capabilities of the workflow management systems (Stohr and Zhao 2001). This framework consists of an engine and five interfaces exposed by the engine, as illustrated in Figure 2.

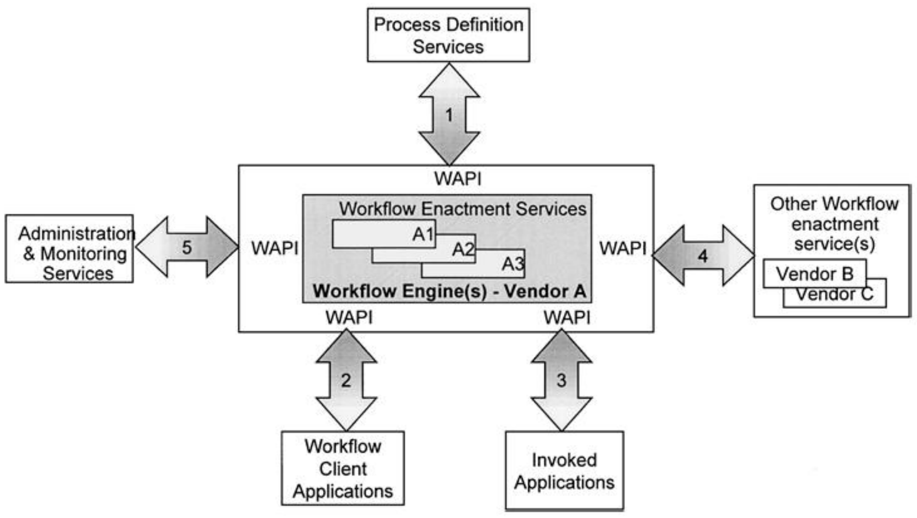

Process Definition Services
1
WAPI
Workflow Enactment Services
A1
A2
A3
Workflow Engine(s) - Vendor A
Administration & Monitoring Services
5
WAPI
WAPI
4
Other Workflow enactment service(s)
Vendor B
Vendor C
WAPI
WAPI
2
3
Workflow Client Applications
Invoked Applications

Figure 2 Workflow standard interface (Stohr, et al. 2001)

The workflow engine provides the core capabilities including:

1) Initiating new workflow instances in response to triggering events;

2) Executing routing logic and determining the human or software agents to perform each of process activities;
3) Driving documents to a selected agent
4) Generating and maintaining a list of tasks (formally named worklist) to be performed by each human or software agent; and
5) Maintaining security and logging all activities.

These interfaces are means for the engine to interact with the external world. Table 6 presents the brief explanation of five standard workflow application program interfaces (WAPIs).

Table 6 Workflow application program interfaces (Stohr, et al. 2001)

	Interface	Remarks
1	Process Definition Services	It is for build-time use to define the workflow process. Usually consists of a graphic interface through which the developer defines the workflow process as a partial ordering of distinct activities
2	Workflow Client Applications	It defines the standard mechanism for interacting with the users of a WfMS - the worklists that appear on user screens, and so on.
3	Invoked Applications	It is an interface for enterprise integration. Through this interface WfMS can interact with user applications, such as ERP or other legacy applications.
4	Other workflow enactment services	This interface enables WfMS provided by different vendors to interoperate. This functionality is of particular importance in e-commerce applications.
5	Administration and Monitoring Services	This interface is for administrators to gather information from the log maintained by the WfMS. This supports managerial control through detailed analysis of the activities of each agent and the performance of the overall workflow process.

It can be seen that a WfMS consists of two essential components: 1) a workflow modeling component; and 2) a workflow enactment component. The former offers a build-time environment where workflow logic can be defined, analyzed and managed. The output of this component is workflow models. The latter, however, provides a runtime environment for the creation, execution and management of workflow instances based on workflow models defined via the modeling component. In the course of workflow execution, the component possibly interacts with actors or some external applications through the five interfaces. The workflow modeling component is a key to the flexibility of WfMS. To achieve the automation of the business process, workflow models need to address the business process from five perspectives, as illustrated in Table 7 (Curtis, Kellner and Over 1992, Jablonski and Bussler 1996).

Table 7 Five perspectives of the workflow model (Stohr, et al. 2001)

Perspective		Remarks
Functional	What does the workflow do?	This perspective specifies the workflow by decomposing high level functions into tasks that can be allocated to human or software agents.
Behavioral	When are the activities and tasks executed?	This perspective defines the time precedence of individual process activities, the events and triggers, and the pre- and post-conditions for activities. Rules associate agents with roles, roles with activities, and activities with data and software applications.
Informational	What data is consumed and produced	This perspective describes the business data, documents, and electronic forms that are transported between agents, and the files and databases that store persistent application information.
Operational	How is a workflow activity implemented	This perspective specifies the workflow tools and applications that perform the discrete steps of the process.
Organizational	Who performs what tasks and with what tools?	This perspective defines the organizational hierarchy, the “roles”, the security and access authorizations, the document approval levels, the teams and work groups that need to be recognized, and the list of agents (individual people and software applications).

2.7.2.2 Research on Workflow Management

In the area of workflow management, workflow modeling, analysis and verification, and workflow change are active research topics. Workflow modeling has been extensively studied. Most modeling techniques are based on Petri nets and graph reduction (Qiu and Wong 2007). Petri nets are a class of modeling tools, which were originated by Petri (Salimifard and Wright 2001). Petri nets have a well-defined mathematical foundation and an easy-to-understand graphical feature. The strong mathematical formalism makes Petri nets possible in describing the behavior of a system by mathematical models. The graphical nature makes Petri nets self-documenting and a powerful design tool. Graph reduction implements an efficient version of non-strict evaluation, an evaluation strategy where the arguments to a function are not evaluated immediately (Wallace, Schimpf, Shen and Harvey 2004).

The foundation of the workflow model lies in its structural specifications. A structural specification may contain conflicts, such as deadlock and lack of synchronization. Identification of such conflicts is a computationally complex process and requires development of effective algorithms specific for the target modeling language. Sadiq and Orlowska (Sadiq and Orlowska 2000) proposed a technique based on graph reduction to identify structural conflicts in workflow models. Verification was also addressed at a conceptual level to identify fundamental problems in workflow specifications (ter Hofstede, Orlowska and Rajapakse 1998). Li et al identified the problem of resource constraints in workflow specifications. They developed corresponding algorithms to verify resource consistency against workflow specifications. The algorithms were extended to timed workflow specifications, where time information is taken into consideration when checking resource consistency (Li, Yang and Chen 2004).

Dynamic adaptability has become one of main features of workflow management systems. There are potential problems in adjusting a workflow process, such as deadlock, inconsistency and even loss of instance. Casati et al proposed a method to facilitate changing workflow schemata by applying a complete, minimal and consistent set of modification primitives (Casati, Ceri, Pernici and Pozzi 1998). Van der Aalst proposed a concept of workflow inheritance to handle dynamic workflow change (van der Aalst and Basten 2002). Qiu and Wong studied dynamics of workflow in PDM systems and developed an approach to facilitate dynamic workflow changes by minimizing repetitive execution of finished workflow nodes. This approach also addressed a data integrity issue by managing various workflow data such as node properties and scripts (Qiu, et al. 2007). Sun and Jiang conducted research on an algorithm to calculate the minimal region affected by workflow structural changes; to check the compatibility of those changes to the original workflow; and to determine whether an active workflow instance would be smoothly evolved to the new workflow (Sun and Jiang 2009).

2.7.2.3 Research on Flexibility of Workflow Management

The concept of workflow flexibility has many interrelated meanings, such as easy design and change, easy enactment of changes in running workflow instances, good support of exception handling and failure recovery and dynamic workflow schema evolution (Agostini and Michelis 2000). Of them, enactment flexibility is most important (Hu and Grefen 2003). Enactment flexibility means that different instances of an activity can be dynamically bound to different implementations at runtime. The conceptual architecture of many WfMSs exhibits flexibility in dynamically binding activity instances with actors by using roles. However, full enactment flexibility has not been yet achieved in most current WFMSs. By addressing enactment flexibility, Hu and Grefen proposed a service-oriented approach to realize dynamically establishing or changing the association between an activity and its implementation. The key point of the approach is to separate the activity specification from the workflow specification. At runtime, the activity specification is dynamically combined with the descriptions of applications or services. Whittingham proposed an Open Water approach (Whittingham 1999), which enables workflow participants to define and execute workflow processes themselves instead of being controlled by a central authority. In this approach, workflow artifacts are passed along from one participant to next, who will need to execute his/her task based on the procedure defined by the previous participants. This is continued until the workflow finishes. Along the time line, the sequence of tasks is captured and stored in an organizational database. Using intelligent techniques, this database can then suggest the appropriate sequence of steps to be followed in the next time when the same workflow is to be executed.

As a summary, workflow management is a process of electronic scheduling and delivery to get the right piece of work to the right person at the right time (Anonymous 2004). It aims to automate the movement of documents (electronic or paper-based) and to ensure that document delivery is progressed in the most efficient way. Different types of flexibility are discussed in the context of workflow management (Heinl, et al. 1999). However, progress is likely to be steady rather than revolutionary because of the technical complexity of workflow management (Stohr, et al. 2001).

2.8 Summary of Literature Review

Research on the flexibility of enterprise systems has been conducted at different levels and from different perspectives. Strategic research provides enterprises with the awareness of business flexibility. Tactical research develops systematic methods and efficient tools for enterprises to represent, analyze and redesign business processes. These methods and tools help enterprises smoothly step through organizational restructuring and make enterprises ready for internal and external changes. However, business flexibility will not come true if enterprise systems cannot be rapidly changed to support changes. SOA and Web services are recent innovative technologies invented for this purpose. They look into the flexibility of enterprise systems from the perspective of interpretability. The flexibility they can provide is limited to functionality sharing and information exchange. These technologies do not address methods for developing flexible enterprise systems.

MDA puts focus on the development of enterprise systems. It is an effort to synchronize business requirements with system source code through automated system generations. However, the synchronization achieved is between system models and system source code. In other words, system models in MDA drive system code generation but not the execution of software programs. To reflect new requirements, redevelopment and redeployment are necessary. Workflow management is a technology closely related to the development of flexible enterprise systems. It enables the behavior of enterprise systems to be adjusted by changing workflow models. However, workflow systems are intended for flow control rather than transactional systems, which involve comprehensive business logic for processing information.

This research looks into the flexibility of enterprise systems from the software development perspective. To extract system models from software programs, a concept of model driven enterprise systems is initiated by leveraging the synergy of MDA and workflow management. In a model driven enterprise system, system models act as instructors to guide and control behaviors of software programs. Software programs function by interpreting instructions in system models. Such a system can be tailored for different enterprises by changing system models. Then, various semantic representations are developed to represent entities, relationships and business logic to support model driven enterprise systems. The concept of model driven enterprise systems and semantic representations forms a systematic method for developing configurable enterprise systems with high flexibility.

Chapter 3 Concept of Model Driven Enterprise Systems

3.1 Introduction

This chapter studies technologies for flexibility in the areas of modern machines and workflow management. The evolution path of system modeling and the role of system models in software development are also explored. By applying analogical reasoning, the concept of model driven enterprise systems is proposed. An abstraction model of enterprise systems is presented for identifying models to be represented in a semantic manner. A reference architecture for model driven enterprise systems is briefly discussed.

3.2 Review of Technologies for Machine Flexibility

In manufacturing, modern computer numerical control (CNC) machines are flexible. They can work for different companies and can produce different shapes of parts. Technologies for machine flexibility have a long evolution history which can be recalled back to the 1800s. Thomas Blanchard realized his gun-stock-copying lathes based on cams between 1820s and 1830s. Christopher Miner Spencer used cams to transform the turret lathe into the screw machine in the 1870s. The cam-based machine flexibility reached a highly advanced state in 1910. In nature, the cam-based method for machine flexibility is to encode engineering drawing information into cams which are then used to control the movement of the machine spindle or cutting tool. Cams are not abstractly programmable. Encoding drawing information into cams is a manual process which requires sculpting and/or machining and filing (Olexa 2001).

Forms of abstractly programmable control appeared in the 1800s (Olexa 2001). Numerical control (NC) for machine flexibility did not eventuate until 1950s. One barrier to NC machines was the tolerances required in the machining process. Although connecting some sort of control to a storage device, such as punched cards, is easy, it is difficult to ensure that the controls are moved to the correct position with the required accuracy.

In the late 1818, Eli Whitney invented a milling machine in New Haven Connecticut. The spindle of Whitney's milling machine stayed vertical rather horizontal as the spindle of other machines stayed. In the early 1930s, Bridgeport, a machine manufacturer, revolutionized Whitney's milling machine with a revolving turret that could move the workpiece in the x, y, and z directions by increments of 0.001 inch by turning the appropriate hand crank. This machine needed to be operated by very skilled operators. By that time, closed-loop control systems, another key technology to the NC machine, became mature. In the 1940s, John Parsons started attaching servomotors to the x and y axis and attempting to control the movement of the machine spindle with a computer. The computer provided servomotors with positioning instructions by reading punch cards. Parsons filed for a patent on "Motor Controlled Apparatus for Positioning Machine Tools" on 5 May 1952 and the NC machine was born. When early servomechanisms were rapidly augmented with analog and digital computers, modern CNC machine tools were developed.

The CNC machine can produce more complicated parts and be easily reprogrammed to produce different parts; thus offers better flexibility. Relatively, the cam based flexibility is limited. From the perspective of the level of flexibility achieved, the cam based method and the CNC based method are different. However, if we think carefully about how machine flexibility is achieved, we can conclude that machine flexibility comes from neither cams nor CNC controller. In fact, the real origination of machine flexibility is the separation of the controller from the machine itself. Such separation enables the controller to stay independent of the machine. Consequently, the controller can be changed or reprogrammed to send out different positioning instructions and the spindle or worktable of the machine can move according to positioning instructions.

By applying analogical reasoning, we can conclude that high flexibility can be achieved in the enterprise system if the enterprise system can be separated into two components like modern machines, as shown in Figure 3. Ideally, in such an enterprise system, the controller stays outside of software programs and issues instructions to guide software programs to perform information processing. A key question raised here is what can be extracted from enterprise systems as the controller. The following section attempts to derive an answer to this question.

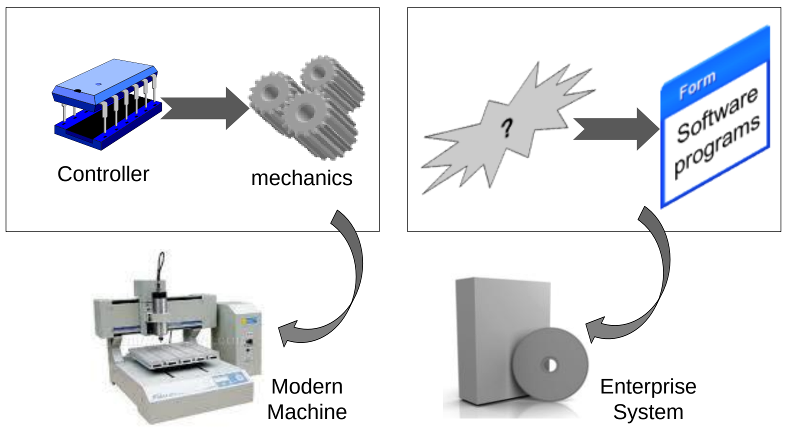


Figure 3 A ideal structure for flexible enterprise systems

3.3 Mechanism for Flexibility of Workflow Management Systems

As discussed in the literature review, workflow is a technology for achieving electronic scheduling and managing document delivery process. It is closely associated with document management and used to drive documents according to the business process. It ensures that the right piece of work goes to the right person at the right time. Support and management of workflow processes in an enterprise is a constant challenge which comes with two contradictory needs: on the one hand, the need for control; and on the other hand, the need for flexibility so that the workflow processes can be adapted easily and quickly to meet constantly changing business conditions (Narendra 2004). Moreover, the workflow process varies from one company to another. In order to make the workflow management system generic enough and able to be changed easily, WfMC defined a standard workflow framework which consists of two components: a modeling component; and 2) a workflow enactment component – workflow engine. Accordingly, two stages are involved in using a WfMS (Whittingham 1999) which are build-time and runtime. At the build-time stage, the workflow process must be understood, captured and represented based on the workflow specification as a computerized model. Once a workflow model is constructed and evaluated, it is then ready for use. At the runtime stage, the workflow engine initiates an instance of the workflow model and starts driving the workflow process against the workflow model.

Compared to the modern machine, WfMS has a similar structure to the one the modern machine has. In WfMS, the workflow model is equivalent to the machine controller and the workflow engine is machine mechanics. A conclusion can be drawn that the flexibility of WfMS also originates from the separation of the workflow model from the workflow engine. The workflow model exists independently of the workflow engine. The workflow model acts as an instructor and the workflow engine as an executive. A workflow engine works by interpreting the workflow model to drive the workflow process. As the workflow model stays outside of the workflow engine, it can be easily changed to make the workflow engine work in different ways.

Workflow management systems are one type of enterprise systems which are specifically used for managing workflows. Unfortunately, they are not sufficient for transactional information management which involves complicated information processing, such as transaction, transformation and association. However, the system structure of modern machines and workflow management can be transplanted to general enterprise systems for transactional information management. This structure provides a clue to answer to the question raised in the previous section. By applying analogical reasoning, it can be concluded that system models need to be extracted and made independent of the software programs in order to achieve highly flexible enterprise systems. The following sections study how system models are related to software programs in current enterprise systems, followed by the initiation of the concept of model driven enterprise systems.

3.4 Unstructured Model Representation

At the initial stage of software development, a majority of software developers took a code-only approach and did not use models defined separately (Brown 2004).

Such an approach was adequate for individuals or very small teams. Due to the lack of documentation, this approach has often led to difficulty in controlling and understanding how business logic is implemented. In turn, it resulted in the difficulty to manage the evolution of systems while the scale and complexity of software systems increased over time. The maintenance of such a system was much harder, especially when it was done by a different team.

Consequently, SDLC management was introduced to ensure that business requirements are documented and systems to be developed are thoroughly analyzed before coding. As illustrated in Figure 4, a typical SDLC consists of multiple phases, including requirement gathering, system analysis, design, coding, testing and deployment. Requirement gathering is a process to understand business concepts, business rules and functions required. The outcomes of this phase are various unstructured documents. The analysis and design phases describe system architecture, functional components and business logic by using various types of documents. Then, these documents are used to guide developers to write system source code. From the modeling perspective, these documents can be deemed as a primitive form of unstructured system models.

Quite quickly the shortcomings of unstructured documents are recognized. Unstructured documents are full of ambiguity and often cause much misunderstanding, and as such, hardly play the role as expected. Standard methods and tools for creating structured representations of business requirements become imperative. Computer-aided software engineering (CASE) was one prominent effort of the 1980s with regard to developing methods and tools for requirements and design documentation. CASE had two major objectives: 1) to enable analyzers and designers to express their design decisions in a graphical fashion, such as state machines, structure diagrams, and dataflow diagrams; 2) to synthesize implementation artifacts from graphical representations to software code (Schmidt 2006). CASE introduced standard graphical diagrams for depicting software architecture, data entities, entity relationships and design decisions. Though graphical diagrams are not well structured models, they are concise and have consistent understanding. They help minimize misunderstanding and effectively guide the creation and evolution of software systems. However, CASE didn't generate significant impacts because these graphical diagrams still played the role of reference. CASE did not establish direct linkages between diagrams and software programs. The referential connection between diagrams and software programs faded away rapidly as the code phase progressed. Designers were not motivated enough to put much effort into the accuracy of diagrams. Consequently, these graphical diagrams were rarely in synchronization with system code in the latter stages of system development. As a whole, CASE contributed to graphical representations of business requirements and business logics, but the synchronization between system models and software code was not successfully achieved.

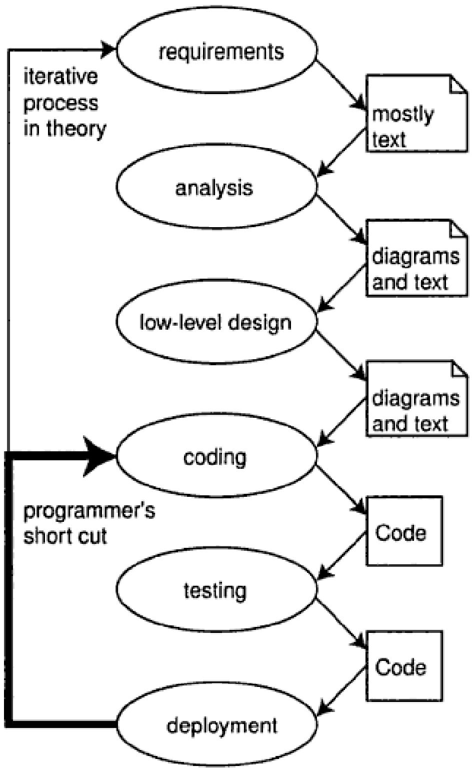


Figure 4 Typical lifecycle of software development (Brown 2004)

3.5 Structured Models in a Formal Language

MDA appears as a promising solution to the synchronization between system models and software source code. MDA employs a formal language, like UML, to document business requirements and design decisions as well structured models. Structured models enable common understanding to be achieved between stakeholders. The formalism of modeling language provides the opportunity to verify system models using various techniques such as Petri nets and graph reduction. More important is a revolutionary change to the role of system models in software development. Instead of being references for guiding developers, system models are used to generate platform specific models. As shown in Figure 5, platform independent models are transformed to platform specific models through system generation tools. Platform specific models are platform dependent system source code. Obviously, system source code gets synchronized with the platform independent model through system generation and regeneration if changes are made to platform specific models. Through generation and regeneration, MDA establishes direct linkage between business requirements and system source code.

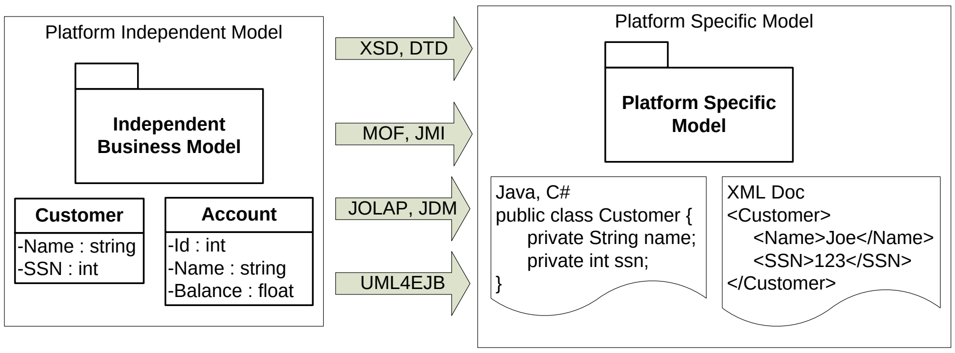


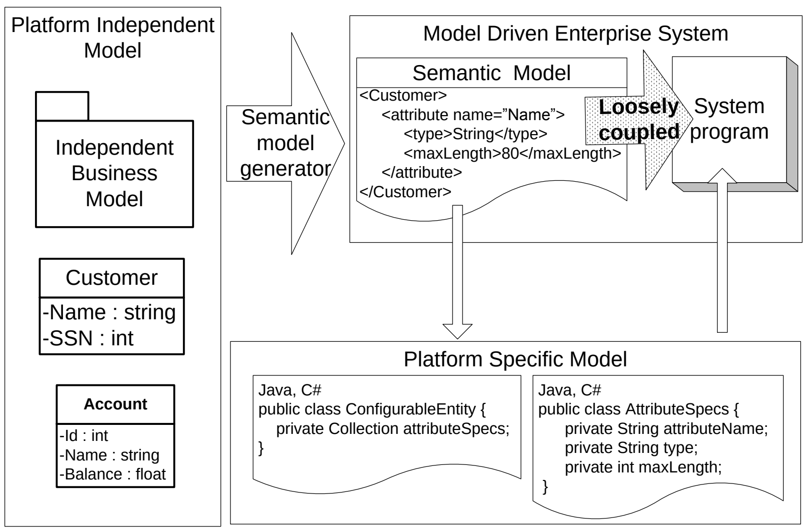
Figure 5 Transformation of PIM to PSM (Brown 2004)

Fundamentally, both CASE and MDA attempts to synchronize business requirements as well as design decisions with system source code. However, they do not prevent hard coding system models in software programs. Software developers digest system models and construct system source code. Throughout the software coding process, system models dissolve into software programs and become intangible. In such a way, software programs work by following procedures predefined by designers and developers. Because system models cannot exist independently outside of software programs, no opportunity exists to control the behavior of software programs by changing system models. Business requirement changes inevitably result in changes to system source code.

3.6 Concept of Model Driven Enterprise System

As analyzed above, enterprise systems can be highly flexible when system models are separated from enterprise systems as an independent component. By taking the advantage of MDA and leveraging MDA model transformation, this research proposes to transform PIMs to neutral models represented in a semantic fashion. Such neutral models represented in a semantic fashion are called semantic models. Semantic models can be used to generate other artifacts of enterprise systems, such as database schema, entity classes and relationship classes. Furthermore, software programs are created to work by interpreting semantic models. In this way, semantic models can be replaced for a different enterprise without changing software programs. When business requirements are changed, semantic models can be revised to adjust the behavior of enterprise systems. Such an enterprise system is referred to as a model driven enterprise system. Model driven enterprise systems offer high flexibility. They can be individualized for different enterprises by providing different sets of system models and be adapted by revising system models for business changes.

The concept of model driven enterprise systems is the convergence of workflow management and MDA and leverages the synergy of workflow management and MDA. As shown in Figure 6, the concept of model driven enterprise systems retains the structure of WfMS to decouple system models from software programs. At the same time, it inherits the concepts of platform independent models and model transformation from MDA. The concept of model driven enterprise systems differs from MDA in that system models can exist independent of software programs. It differs from WfMS in model representations. It is obvious that model representations are a critical factor to the success of mode driven enterprise systems. To really make a model driven enterprise system flexible, system models should be represented in a format which is easy-to-understand and easily constructed by human beings. Meanwhile, they should also be able to be effectively interpreted by computers. The following sections discuss what system models are and how system models can be fitted into model driven enterprise systems. Semantic representations will be investigated in the next chapter.



```
<Customer>
    <attribute name="Name">
        <type>String</type>
        <maxLength>80</maxLength>
    </attribute>
</Customer>
```

```
Java, C#
public class ConfigurableEntity {
    private Collection attributeSpecs;
}
```

```
Java, C#
public class AttributeSpecs {
      private String attributeName;
      private String type;
      private int maxLength;
}
```

Figure 6 Concept of model driven enterprise systems

3.7 Abstraction of Enterprise Systems

Enterprise systems support business operations by managing various information entities and associations between information entities. From this perspective, an abstraction model of enterprise systems can be represented as Figure 7. Fundamentally, six key elements can be identified from the abstraction model: 1) information entities; 2) entity relationships; 3) functions; 4) process flows; 5) working environment; and 6) security management. Information entities represent business objects (physical items, such as parts and products or virtual items, such as projects).

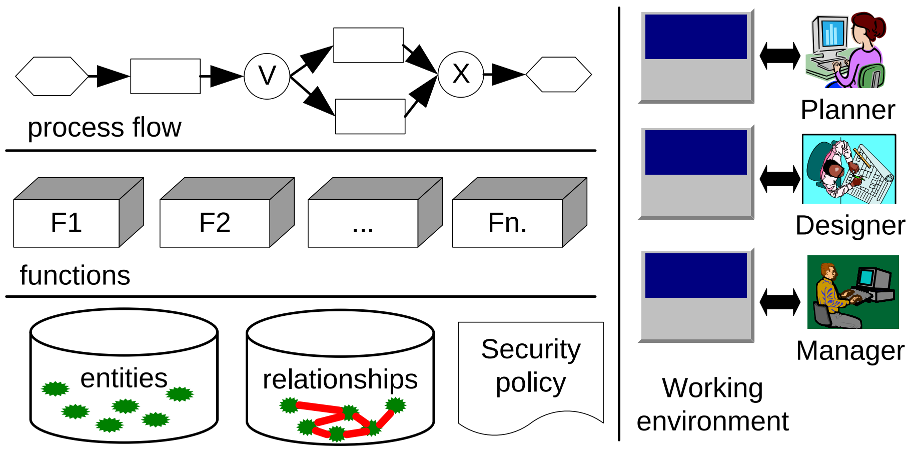


Figure 7 Abstraction model of enterprise systems

In general, entities are business concepts and terms; and entity relationships are the projections of business practices. Throughout business processes, various information entities are accumulatively created, manipulated and associated to support or drive continuing business activities. Enterprise systems expose various functions to support business activities by following business process flow. Behind those functions, enterprise systems need to effectively manage information entities and associate information entities to achieve continuous information flows. Functions offered by enterprise systems should be logically organized in accordance with process flow, and clustered into various groups for users with different roles. To enable users to effectively interact with enterprise systems, efficient working environments should be presented to end users to access functions offered by enterprise systems.

The development of enterprise systems basically involves: 1) capturing business requirements; 2) establishing system models; and 3) developing functions for users to manipulate information. System models need to be built by correctly understanding business requirements and concepts. Functions are realized by creating software programs. Functions have to be developed according to system models which represent the need of business operations. In enterprise systems, security needs to be well managed to control access to information and functions.

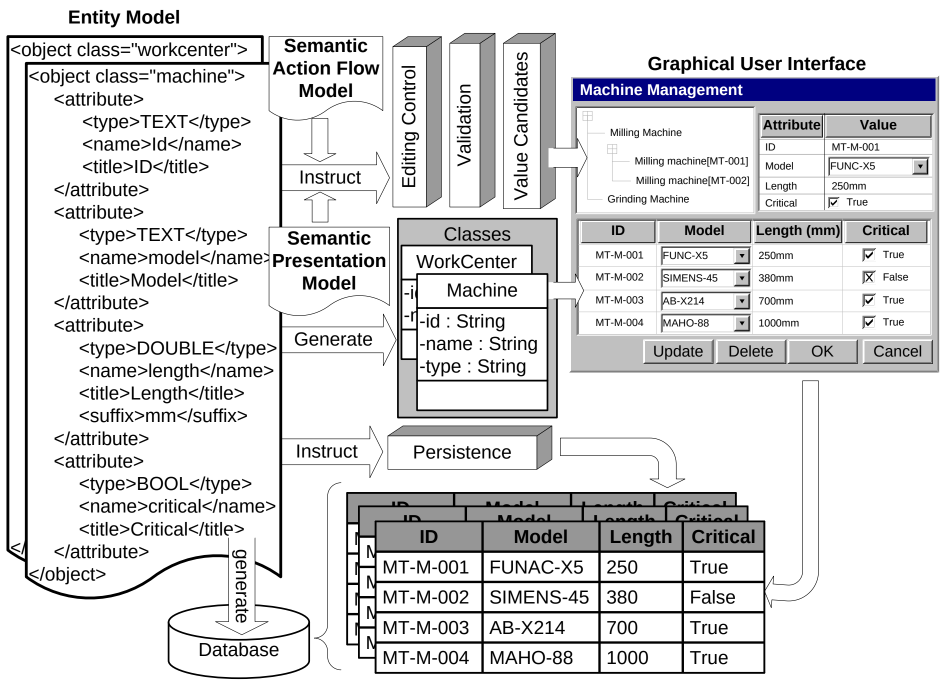
Therefore, to achieve a really flexible enterprise system, it is essential to extract the following models from enterprise systems: 1) entity model; 2) relationship model; 3) processing logic model; 4) scenario model; 5) function layout model; and 6) graphic user interface (GUI) presentation.

3.8 Reference Architecture for Model Driven Enterprise System

Architecture, which conceptually characterizes a software system, is a critical factor decisive to the success of application development and integration. Architecture design is a process used to determine system structure, define core components and identify common functions. From a structural viewpoint, it defines a way for decomposing a system into various interrelated building blocks (Doumeingts, Ducq, Vallespir and Kleinhans 2000). From a functional viewpoint, it identifies useful design patterns and supports the patterns by providing abstract layers (Gamma, Helm, Johnson and Vlissides 1998). Design patterns provide unified approaches to solving similar problems and make system development more efficient. Therefore, architecture design is a key step in the development of enterprise systems. A well-designed architecture provides a solid common foundation and enables the focus of system development on domain functions with less work on fundamental functions.

With these concerns incorporated, the architecture for model driven enterprise systems is developed as shown in Figure 8. The proposed architecture provides an effective mechanism for incorporating various semantic representations to increase the flexible of enterprise systems. Semantic representations are categorized as follows: 1) semantic entity representation; 2) semantic entity relationship representation; 3) semantic logic representations, such as editing control, value validation and deriving value candidates; and 4) semantic environment representation. Entity representation and entity relationship representation are the foundations of other representations. They are also used to generate database schema and entity classes and to guide data persistence. Logic representations are intended for describing logic to control information processing, such as data validation. Environment representations contain instructions to guide GUI layout, information presentation and function layout.

Taking machine management as an example, the machine entity model is semantically represented in XML. This semantic model is utilized to generate machine table schema and machine entity class. Data persistence is also to be completed by interpreting the semantic machine model. The presentation of machine attributes is carried out by dynamically interpreting the semantic model at runtime. The semantic model plays a critical role in the synchronization of the database table, entity class, persistence and presentation. To individualize the machine management function, the machine entity model can be constructed or reconstructed based on the specific needs of a company. Likewise, logic represented in a semantic manner can also be easily changed to have a different method to control editing process, a different validation rule and a different set of value candidates. The machine database table and machine entity class can be regenerated based on the updated entity models. Since presentation and persistence are performed dynamically by interpreting the entity model at runtime, they do not have to be changed. Therefore, such a system can be rapidly configured and reconfigured for different enterprises.



```
<object class="workcenter">
  <object class="machine">
    <attribute>
      <type>TEXT</type>
      <name>Id</name>
      <title>ID</title>
    </attribute>
    <attribute>
      <type>TEXT</type>
      <name>model</name>
      <title>Model</title>
    </attribute>
    <attribute>
      <type>DOUBLE</type>
      <name>length</name>
      <title>Length</title>
      <suffix>mm</suffix>
    </attribute>
    <attribute>
      <type>BOOL</type>
      <name>critical</name>
      <title>Critical</title>
    </attribute>
</object>
```

```
-id : String
-name : String
-type : String
```

Attribute	Value
ID	MT-M-001
Model	FUNC-X5
Length	250mm
Critical	✓ True

ID	Model	Length (mm)	Critical
MT-M-001	FUNC-X5	250mm	✓ True
MT-M-002	SIMENS-45	380mm	✗ False
MT-M-003	AB-X214	700mm	✓ True
MT-M-004	MAHO-88	1000mm	✓ True

ID	Model	Length	Critical
MT-M-001	FUNAC-X5	250	True
MT-M-002	SIMENS-45	380	False
MT-M-003	AB-X214	700	True
MT-M-004	MAHO-88	1000	True

Figure 8 Model-driven system concept

3.9 Summary

This chapter transplants the structure for flexibility in the area of modern machines and workflow management to enterprise systems; and derives the concept of model driven enterprise systems by applying the analogical reasoning method. System models to be extracted from enterprise systems are identified based on the abstraction model of enterprise systems. A reference architecture is proposed for the incorporation of semantic models to model driven enterprise systems. The next chapter develops semantic representations of various system models. According to the concept initiated, an enterprise system consists of two essential components: semantic models and software programs. Semantic models serve as instructors to guide and control the execution of system programs. In turn, software programs act as executives to carry out processing by interpreting instructions from semantic models. When models are changed or replaced, software programs can accomplish processing according to new models. The customization of a model driven enterprise system is to configure semantic models. The adaption of a model driven enterprise system is to reconfigure related semantic models. The flexibility of model-driven enterprise systems is achieved through model configuration and reconfiguration.

Chapter 4 Semantic Model Representations

4.1 Introduction

Model representations are critical in modeling driven enterprise systems. To enable model driven enterprise systems to be configured and reconfigured, model representations need to be easily understood by human beings while they can be effectively processed by computers. XML has been a standard markup language and has been widely adopted for metadata representations. A variety of tools targeted for different platforms are available to process XML documents. XML fulfills the needs of semantic representations. Therefore, this chapter develops semantic representations for various system models by using XML.

4.2 Semantic Entity Representation

The attributes of the same business entity can vary from one company to another due to the difference of business practices. A model driven enterprise system needs the ability to introduce new entities, add attributes to an existing entity or remove, visually if not physically, attributes which are not required. This research reveals essential techniques that enable the introduction of new entities and changing attributes of an existing entity, including:

1) entities can be declared in a standalone neutral file;
2) new attributes can be added to an entity;
3) unrequired attributes can be clearly indicated;
4) a mechanism built into entity classes to hold the values of extended attributes;

5) processing logic can be injected for validating the values of extended attributes;
6) the values of extended attributes can be automatically stored to database; and
7) an adaptive environment presentation is needed to present extended attributes or hide unwanted attributes.

By taking these needs into consideration, an entity declaration model, as illustrated in Figure 9, is developed for declaring entities in a neutral format outside of software programs. This model provides a means to semantically define entities and attribute specifications. A semantic entity declaration comprises a set of semantic attribute definitions. A semantic attribute definition depicts basic information about an attribute, such as attribute name, type, title and editor. In Figure 9, *Properties* is introduced into attribute definition for describing the miscellaneous characteristics of attributes, such as *editable, visible*, *title suffix,* and *transient* etc. A title suffix is a short text to be attached to a title for display, such as the unit name. Transient attributes are not to be persisted to database. Miscellaneous information can be used to manage GUI presentation, control editing and guide data persistence.

As shown in the model, an attribute can be modeled, extended or category dependent. A modeled attribute is an attribute designed at the development stage. An extended attribute is the one added after a system is developed. A category dependent attribute is the one an entity has, only when the entity is associated with a category. For example, a *Resource* class is designed as a generic representation of resources. Attributes modeled in the *Resource* class are common to all categories of resources. Machines and cutting tools are two specific categories of resources. Each category needs a specific set of attributes. For example, the machine needs two additional attributes: *model* and *spindleSpeed*; and the cutting tool needs another two different attributes: *material* and *shankDiameter*. A semantic category model can be defined for machines and cutting tools respectively. Each category model has its corresponding two attributes defined. When initiating an instance of the *Resource* class, it needs to be associated with a category. If it is associated with the machine category, two attributes defined in the machine category model are dynamically incorporated into the instance. Likewise, if it is associated with the cutting tool category, two attributes defined in the cutting tool category are incorporated into the instance.

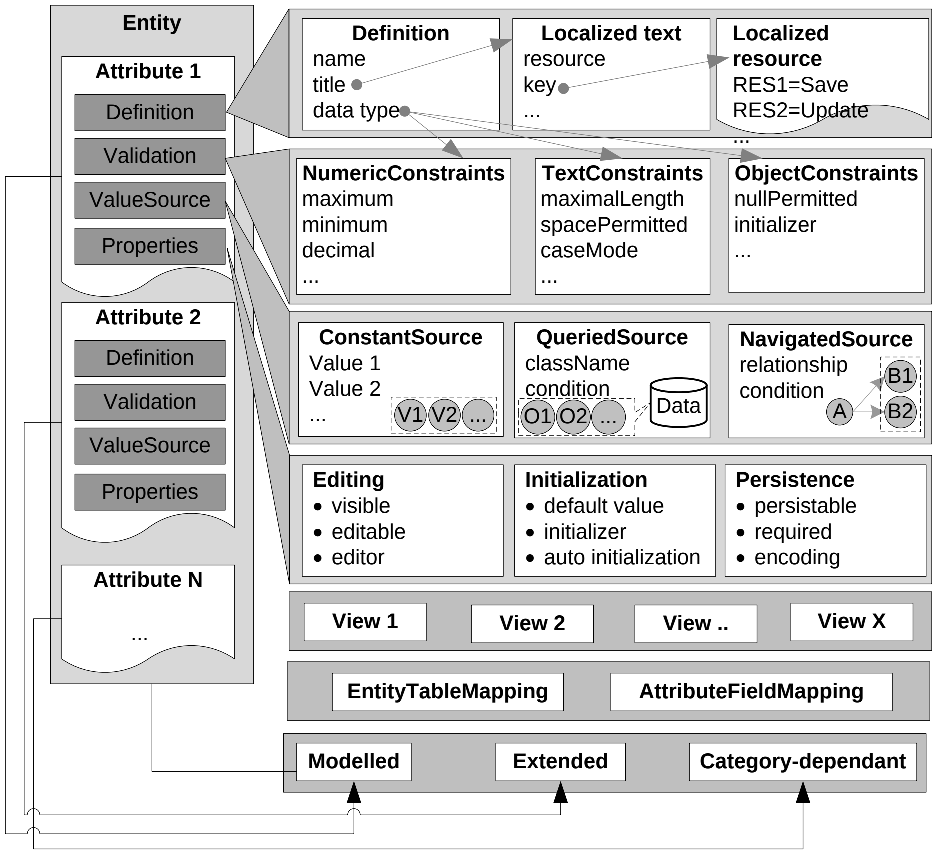


Figure 9 Overview of semantic entity representation

The entity declaration model supports the concept of view. An entity view consists of a subset of attributes. The entity declaration model enables an entity to have multiple views associated with it. The view declaration can redefine default value, value source, and value validation to override counterparts defined in entity declarations. Views are mainly defined for different roles.

The model also enables an attribute to be further characterized by defining value constraints, value validation, value source, mapping between entities to database tables and default value. Value constraints are simple rules that can be attached to attribute editor for validating user input. Value constraints should be defined in accordance with data types. Legal constraints for the text type include: 1) maximum length of value; 2) uppercase or lowercase only; and 3) whether spaces are permitted. Legal constraints for the numerical data can be: 1) maximum value; 2) minimum value; and 3) the number of decimal digits. Value constraints can also be used in database schema generation.

More complicated validation logic can be defined in value validation. A value validation is a logical expression representing complicated validation logic. As shown in Figure 10, given that the length of bolts is numerical but it is standardized, it cannot be any numerical values. In the expression, “it” represents the value of the attribute the expression is attached to. The *indexOf* function returns the index of an element in a list. If the index returned is greater than or is equal to zero, it means that the attribute value falls into the list. If it is less than zero it means that the attribute value does not exist in the list. This expression requires that the length of bolt must be one of the values in the list. Value validation expressions are usually used to validate attribute values based on business rules. Figure 10 indicates that value validation can be limited to some specific actions, such as *save* and *update*. This enables an attribute to have different validation logic associated for different actions. For example, the *delete* action may need different validation logic.

```
<validation class="cbd.enterprise.resource.SampleEntity">
    <attribute name="length">
        <type>error</type>
        <actions>
            <action>save</action>
            <action>update</action>
        </actions>
        <condition>
            <![CDATA[indexOf(it, list(125, 250, 450))>=0]]>
        </condition>
    </attribute>
</validation>
```

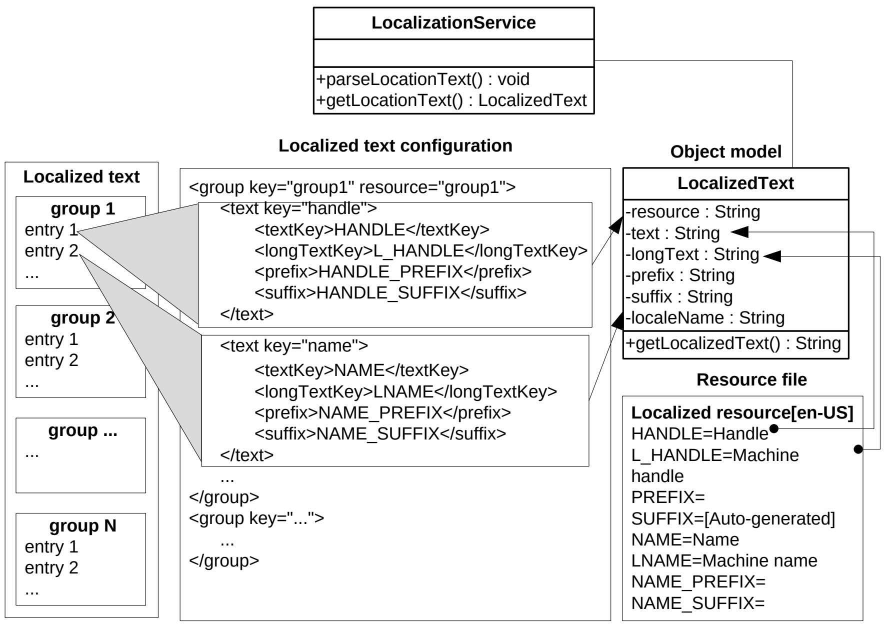
Figure 10 Attribute validation expression

Value sources provide information for deriving the value candidates of attributes. Value candidates can be displayed as a dropdown list for selection in an editing environment. Three types of value sources can be defined, which are constant value source, query value source and navigation value source. The constant value source defines a set of constants as value candidates. The query value source contains information for retrieving value candidates from a database. The navigation value source provides information for deriving value candidates by evaluating an expression.

The table map is used to define the mapping between entities and database tables. The field map is for defining the mapping between entity attributes and table fields. The table mapping and field mapping are used to generate database schema and to guide storing entities to database. To support attribute extension, each entity table needs to have an additional field for storing the values of extended attributes. Two approaches can be used to store the values of extended attributes in a database. One is to use a collection to hold all the values of the extended attributes, serialize the collection as a binary data set and store the binary data set as a binary large object (BLOB). Another approach is to store the extended attribute values as XML fragments. The first approach needs less effort because most computer languages support the BLOB data type. The disadvantage of the first approach is that the binary data can only be reversed back to a corresponding language-dependent object. The second approach is language-independent and some database systems already support the XML data type. This is the preferred approach to store values of extended attributes.

This entity declaration model separates the default value definition and descriptive title configuration from the entity declaration. This separation enables different teams to work collaboratively to construct semantic models. People who have technical skills of system configuration may focus on attribute declarations while people with business domain knowledge can carry out the definition of default values and display titles. In addition, descriptive information may need to be localized. The separation of descriptive information from the entity declaration makes localization independent of the entity declaration. Mixing up descriptive information with the entity declaration potentially leads to unexpected changes to the attribute declaration. The entity declaration model introduces a text resource configuration for defining descriptive texts. As shown in Figure 11, the *text* tag represents a text resource element. The *key* attribute of the *text* tag is to associate this element with an attribute. The *textKey* and *longTextKey* tags are used to link this element to a text element in a resource file. The *LocalizedText* and *LocalizedTextService* class are entity class and service class for managing the localized resource configuration.

A formal declarative language is needed to define entities based on the developed entity semantic representation model. An XML-based entity declaration language, named Attribute Declaration Language (ADL), is developed for this purpose. Table 8 presents commonly used tags as examples.



```
<group key="group1" resource="group1">
    <text key="handle">
        <textKey>HANDLE</textKey>
        <longTextKey>L_HANDLE</longTextKey>
        <prefix>HANDLE_PREFIX</prefix>
        <suffix>HANDLE_SUFFIX</suffix>
    </text>
    <text key="name">
        <textKey>NAME</textKey>
        <longTextKey>LNAME</longTextKey>
        <prefix>NAME_PREFIX</prefix>
        <suffix>NAME_SUFFIX</suffix>
    </text>
    ...
</group>
<group key="...">
    ...
</group>
```

```
Localized resource[en-US]
HANDLE=Handle
L_HANDLE=Machine
handle
PREFIX=
SUFFIX=[Auto-generated]
NAME=Name
LNAME=Machine name
NAME_PREFIX=
NAME_SUFFIX=
```

Figure 11 Text resource configuration

Table 8 Commonly used keywords of descriptive language

Type	Keyword	*Explanation*
Core	name	Defines name of an extended attribute
	type	Defines attribute data type
	extended	Indicates if attribute 's modelled or extended attribute
	defaultValue	Defines the default value of the attribute
Specialization	extends	Specifies parent schema
	deprecated	Indicates that an attribute is deprecated
Display	title	Defines a text as the display title of an attribute
	visible	Indicates whether the attribute is visible on screen
	prefix	Defines a text as the prefix of the attribute title
	suffix	Defines a text as the suffix of the attribute title

Table 8 Commonly used keywords of descriptive language (Continued)

Type	Keyword	*Explanation*
Editing	editable	Indicates whether the value is editable on screen
	required	Indicates whether the attribute is compulsory
	Suffix	Defines a text as the suffix of the attribute title
	valueSource	Define a set of possible values
Validation	maximalValue	Defines a text as the display title of an attribute
	minimalValue	Defines a text as the prefix of the attribute title
	maxLength	Defines a text as the suffix of the attribute title
	decimalNum	Defines the number of digits after the decimal point
	uppercase	Automatically converts characters to uppercase
	lowercase	Automatically converts characters to lowercase
	spacePermitted	Indicates whether spaces in the attribute value is allowed

Figure 12 demonstrates a simplified entity model in ADL. The tag *entity* wraps the declaration of an entity. To avoid the unnecessary duplications of attribute declarations and improve the efficiency of semantic model construction, an entity model can be declared by extending another entity model. The attribute *super* of the tag *entity* is used to specify a super entity. The attribute *class* of the tag *entity* defines the class that represents this entity. The tag *attribute*, a nested tag of the tag *entity*, declares entity attributes. Attribute name, data type and display title are mandatory information in the declaration of attributes. As shown in Figure 12, attribute name is defined by the attribute *name* of the tag *attribute*. Data type and display title are defined by the nested tags *type* and *title*. The tag *extended* is employed to indicate that an attribute is an extended attribute (if the tag value is “true”) or a built-in attribute (if the tag value is “false”). The tag *required* indicates whether the attribute value can be empty when storing the entity to a database. The tag *attribute* can also have a nested tag *deprecated*. This tag can mark an attribute as unwanted by setting the tag value to “true”. An unwanted attribute will be made invisible to end users.

4.3 Semantic Relationship Representation

Relationships between entities are the projections of business practices. Figure 13 shows a model representing relationships. Given class *A* and class *B*, the class

ABLink represents their relationship. The class *ABLink* contains the information which is meaningful only when the association between the two entities exists. For example, the salary information makes sense only when a person is an employee of a company. As such, it should be an attribute of the link class of company and employees. In enterprise systems, relationship models are templates which guide the management to associations between the instances of two interrelated classes. Traditionally, relationships between different entities are hard coded into corresponding management programs. After the system is developed, changes to relationships lead to much rework. In order to make relationships configurable, even after the system is developed, relationships need to be represented in a semantic way.

```
<entity class=“person">
    <attribute name=“firstName”>
        <type>TEXT</type>
        <title>First name</title>
    </attribute>
    <attribute name=“givenName”>
        <type>TEXT</type>
        <title>Given name</title>
    </attribute>
</entity>
<object class=“employee“ super=“Person”>
    <attribute name=“employeeNo “>
        <type>TEXT</type>
        <title>Employee No</title>
    </attribute>
</entity>
<entity class=“supervisor“ super=“employee”>
    <attribute name=“role”>
        <type>TEXT</type>
        <required>true</required>
        <title>Role</title>
    </attribute>
    <attribute name=“workshop”>
        <type>TEXT</type>
        <deprecated>true</deprecated>
        <title>Workshop</title>
    </attribute>
    <attribute name=“remarks”>
        <type>TEXT</type>
        <extended>true</extended>
        <title>Remarks</title>
    </attribute>
</entity>
```

Figure 12   Simplified semantic entity representation

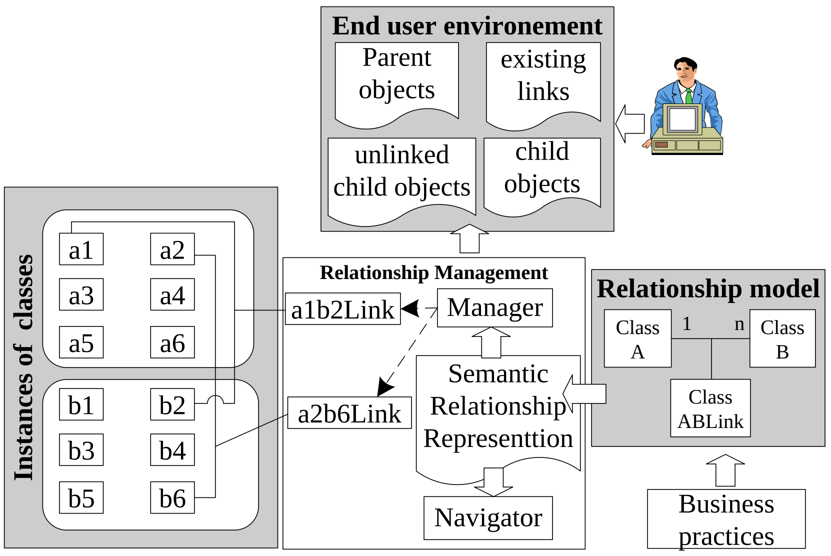


Figure 13 Overview of semantic relationship representation

To achieve flexible relationship management in model driven enterprise systems, relationships between entities need to be represented in a semantic fashion and management programs are implemented based on semantic relationship models. Semantic relationship models stay outside of the programs. Therefore, entity relationships can also be flexibly configured. When the models are changed, the programs can automatically manage relationships based on new models without the need to modify system source code. Relationship management functions can be categorized into two groups: relationship maintenance, such as relationship creation or removal, and relationship navigation, which is to retrieve linked objects or link objects by using a known object based on relationship definitions. Accordingly, semantic relationship models are divided into two categories: relationship definitions and navigation configurations. A relationship definition provides information about how two objects should be associated, such as cardinalities and primary keys.

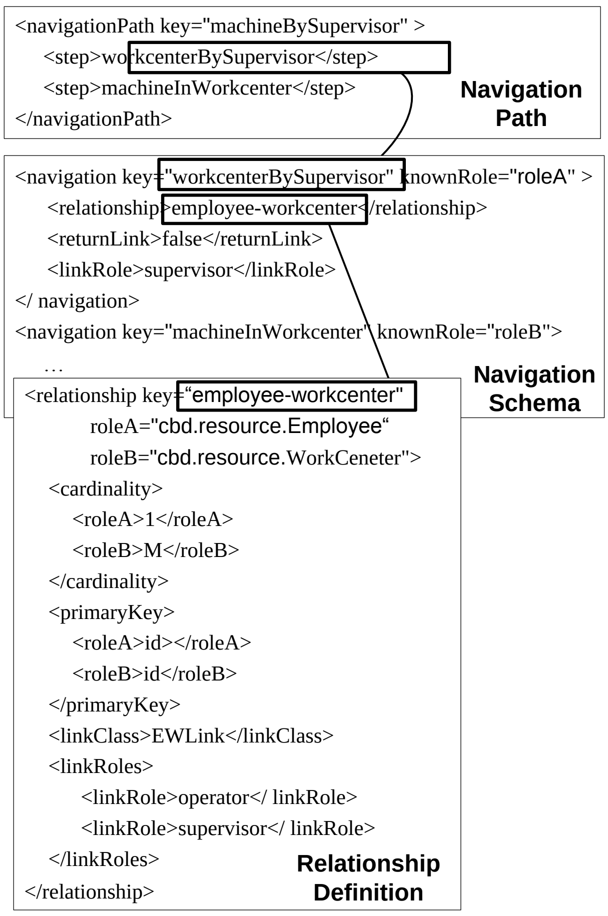
Figure 14 illustrates an example of the semantic relationship model. Two instances of a class can be associated with the same instance of another class for different purposes. For example, an employee can be associated with a work center as a member. Another employee can be associated with the work center as a supervisor. These two different types of associations can be represented as two different relationships by using two link classes. They can also be represented as one relationship by introducing link roles. The aggregation of different types of associations can simplify a management program. To support this flexibility, the tag *linkRole* is introduced for defining the roles of associations. The model also enables one association for multiple roles. The tag *linkRoles* is used for this purpose. The tag *linkRoles* can have multiple nested tag *linkRole*. Other tags in the relationship definition are self-explanatory.

Navigation configurations are further classified into two categories: one-step navigation and multi-step navigation. The one-step navigation is referred to as retrieving linked objects or link objects based on known objects. The multi-step navigation is formed by two or more joint one-step navigations. For example, supervisors have associations with work centers and further work centers are linked with machines. To know the machines managed by a known supervisor, firstly, one or more work centers can be retrieved based on the known supervisor. Then, the machines within each work center can be navigated. Obviously, retrieving machines under a known supervisor consists of two adjacent one-step navigations. As shown in Figure 14, navigation schemata are used to define one-step navigations, and navigation paths are adopted to define multi-step navigation.

4.4 **Semantic Logic Representation**

Enterprise systems often involve various types of logic for processing information, such as transformation, calculation and identity generation. Model driven enterprise systems need to extract logic and represent logic as semantic models. Three types of logic representations are investigated in this research: 1) mapping; 2) pattern; and 3) expression.

Semantic mapping provides simple logic to transform one value to another value, such as transforming a keyword to a descriptive text for display. A simple example is the choices of “*Yes*” and “*No*”. In database, the *Yes* choice may be saved as “*1*” and the “*No*” choice as “*0*”. When presenting these choices on screen, “*Yes*” and “*No*” are more descriptive to end users. Figure 15 illustrates a priority mapping model.



```
<navigationPath key="machineBySupervisor" >
    <step>workcenterBySupervisor</step>
    <step>machineInWorkcenter</step>
</navigationPath>
```

```
<navigation key="workcenterBySupervisor" knownRole="roleA" >
    <relationship>employee-workcenter</relationship>
    <returnLink>false</returnLink>
    <linkRole>supervisor</linkRole>
</ navigation>
<navigation key="machineInWorkcenter" knownRole="roleB">
    …
```

```
<relationship key="employee-workcenter"
          roleA="cbd.resource.Employee"
          roleB="cbd.resource.WorkCeneter">
    <cardinality>
       <roleA>1</roleA>
       <roleB>M</roleB>
    </cardinality>
    <primaryKey>
       <roleA>id></roleA>
       <roleB>id</roleB>
    </primaryKey>
    <linkClass>EWLink</linkClass>
    <linkRoles>
         <linkRole>operator</ linkRole>
         <linkRole>supervisor</ linkRole>
    </linkRoles>
</relationship>
```

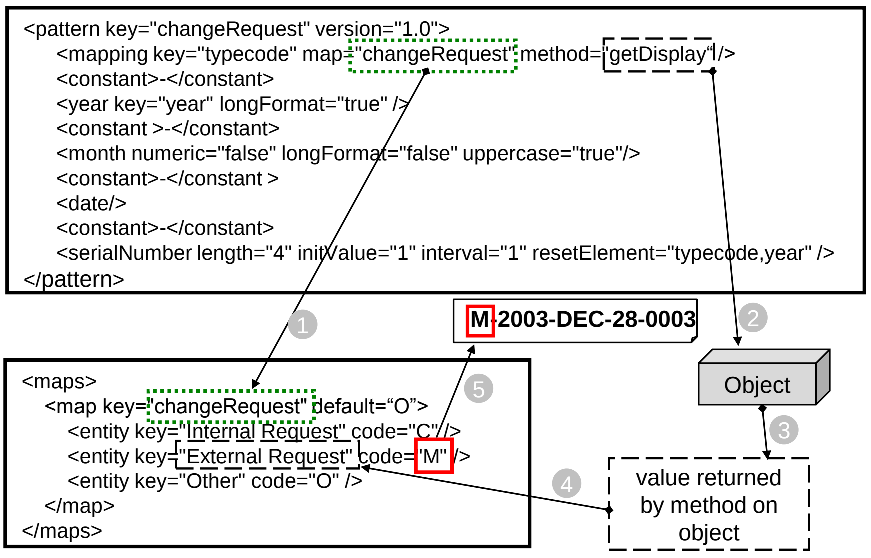
Figure 14   Semantic relationship model

```
<mapping>
      <group key="priority" >
            <map key="High Priority " code="C" />
            <map key="Medium Priority " code="M" />
            <map key="Low Priority " code="L" />
      </group>
</mapping>
```

Figure 15   Semantic mapping

A semantic pattern is developed for representing complicated processing logic. Identity generation is adopted to demonstrate semantic pattern representation. Each business item has an identity to uniquely identify it. The format of identities for different items can be different. It also varies from one company to another. If identity formats are not configurable, much effort is needed to customize an enterprise system for a particular need. A pattern based identity generation can provide a significant advantage. As shown in Figure 16, an identity pattern is developed for generating identities for change requests. The pattern consists of a set of sequential elements. An element can be one of the following types:

- *Constant*, the element is a constant value;
- *Year*, the element is a four or two digital number of the year when that object is created. This tag has an attribute *longFormat*. It is used to indicate a year is represented in a two-digital or four-digital number;
- *Month*, the element can be a two digital number, full name or three-character short name of the month when that object is created. This tag has three attributes: *numerical*, *longFormat* and *uppercase*. The attribute *numerical* indicates using a digital number or the name of months. The attribute *longFormat* indicates using a full name or short name for the month in the case where the tag *numerical* is set to false. The attribute *uppercase* is applicable only when tag *numeric* is set to false. It indicates that month names should be uppercase or lowercase;
- *Date*, the element is a two digital number of the date when that object is created;
- *Mapping*, the element value is to be derived based on a semantic mapping model. This also shows how semantic representations can be integrated to represent complicated business logic;
- *Serial number*, most identities contain a sequential number. The sequential number is continuous length-fixed digits. In this tag, initial value, interval and length can be defined for a sequential number. In addition, a sequential number may need to be reset under certain conditions. For example, a sequential number can be reset every new year. In this tag, multiple dependent elements can be defined. When the dependent element values are different from the previous one, the sequential number will be reset to the initial value. In Figure 16, dependent elements are the first element *mapping* and the third element *year*. It means that each identity starting with "*O*", "*C*" or "*M*" has its own sequential number and the sequential numbers are reset to the initial value each new year.



```
<pattern key="changeRequest" version="1.0">
    <mapping key="typecode" map="changeRequest" method="getDisplay"/>
    <constant>-</constant>
    <year key="year" longFormat="true" />
    <constant >-</constant>
    <month numeric="false" longFormat="false" uppercase="true"/>
    <constant>-</constant >
    <date/>
    <constant>-</constant>
    <serialNumber length="4" initValue="1" interval="1" resetElement="typecode,year" />
</pattern>
```

```
<maps>
  <map key="changeRequest" default="O">
    <entity key="Internal Request" code="C"/>
    <entity key="External Request" code="M"/>
    <entity key="Other" code="O" />
  </map>
</maps>
```

Figure 16 Semantic pattern for identity generation

Obviously, more types of elements can be supported for more complicated processing logic. Semantic pattern representation can also be used for other purposes, such as generating display titles, formatting numbers and deciding images for different types of entities. Figure 17 shows another semantic pattern model for list machines as a table. The pattern indicates that ‘id’ is displayed as a hyperlink and ‘model’ is displayed with an image in front of it.

The expression is a comprehensive approach to represent business logic. An expression is a set of sequential elements linked together by mathematical operators, logical operators and some functions. Expressions can be used to transform values, perform value calculations, and validate values. An element can be constant operand or a call to another expression. In semantic expression representation, the dot is a specific operator representing a method, such as *machine.getTitle()*, or an attribute, such as *machine.model*, of an object. The key on the left of a dot, such as *machine*, is the identifier of an object. The key on the right side is a method if it is followed by a pair of parenthesis, or an attribute if it is not followed by parenthesis. Besides normal mathematic operators and logical operators, various functions can be developed for constructing expressions. Table 9 presents the commonly used functions developed in this research. Figure 18 shows an example expression for calculating the cost of a process step.

```
<conf type="table_pattern">
    <pattern key="cbd.enterprise.resource.Machine"
    title="MACHINE_TABLE_PATTERN">
        <column attribute="Id">
            <template><![CDATA[<a href= /cbd/ScenarioRouter >
            $idValue$</a>]]></template>
            <valueConfig>
                <value key="$idValue$" type="attribute" source="Id"/>
            </valueConfig>
        </column>
        <column attribute="Title">
            <template><![CDATA[$nameValue$]]></template>
            <valueConfig>
                <value key="$nameValue$" type="attribute" source="Title"/>
            </valueConfig>
        </column>
        <column attribute="Model">
            <template><![CDATA[$modelValue$]]></template>
            <valueElement>
                <element>$modelImage$</element>
            </valueElement>
            <valueConfig>
                <value key="$modelValue$" type="attributeValue" source="Model"/>
            </valueConfig>
        </column>
    </pattern>
</conf>
```

Figure 17 Semantic pattern for listing machines

Table 9 Common functions for constructing expressions

Function Name	Remarks
sqrt	Calculate square root
average	Calculate average of a set of values
max	Return the maximal value of a set of values
min	Return the minimal value of a set of values
date	Get current date
toDate	Convert a text to date
month	Get current moth
season	Get current season
round	Round a decimal value
int	Get integer of a decimal value
fraction	Get fraction of a decimal value
switch	Execute a block based on the input value
for	Starts a loop
createArray	Initiate an array
toArray	Convert a collection to an array

Table 9 Common functions for constructing expressions (Continued)

Function Name	Remarks
isArray	Check if an object is the array type
arrayLength	Get the length of an array
indexOf	Return the index of a given element in an array
callMethod	Call a method on an object with parameters
expression	Call another expression
startsWith	Check if a text starts with a given text
contains	Check if a text contains a sub text
endsWith	Check if a text ends with a sub text
isExistingFile	Check if a file physically exists

```
<conf type="expression">
    <expression name="processingCost" title="Processing Cost">
          <formula>
          <![CDATA[
             Machine.costRate * process.processingTime
                       + labor.costRate * process.processingTime
          ]]>
          </formula>
    </expression>
</conf>
```

Figure 18   Expression for calculating processing step cost

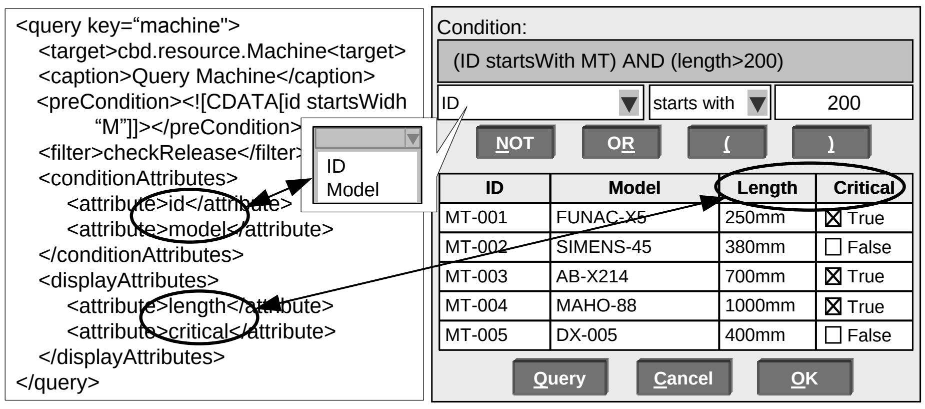
4.5 Semantic GUI Component Representation

Semantic representations can be used to develop some generic components. They can greatly increase the reusability of software components and make enterprise systems more flexible. An object query component based on a semantic query model is used to illustrate this advantage.

In entity management, end users often make queries to retrieve a set of objects for selection. Different objects have different sets of attributes and are stored in different database tables. In general, it is difficult to develop a generic query component for different objects. Semantic representations can be used to develop a generic query component. As shown in Figure 19, a semantic query model is adopted to define information for retrieving and displaying objects. A semantic query model consists of pre-conditions, conditional attributes, display attributes and filters.

The query model divides the query condition into two parts: pre-condition and runtime condition. A pre-condition is the one which is common to all retrievals made, based on the same query. For example, in the creation of reports listing purchase requests in the state of *approval*, the value of the *state* attribute is “approval” and can be deemed as a pre-condition. A runtime condition is one that is constructed by end users at runtime based on GUI(s). In a case where a purchase manager would like to create a report listing under-approval purchase requests raised by the design department, the value of the department attribute is “design” which can deemed as a runtime condition. The pre-condition and runtime conditions are to be combined to form a complete query condition.

In the query model, conditional attributes are the ones that can be used to construct runtime conditions while display attributes are the ones that are not used for constructing runtime conditions, but need to be presented to end users while listing retrieved objects. A filter is an expression which can be used to refine a query result. After a collection of objects are retrieved, each object in the result is used as input to evaluate the filter expression to decide the object should be kept in, or removed from the query result. Filters can be understood as extensible and comprehensive post processors. Based on semantic query models and semantic entity models, such a query component can be used to query different types of objects.



```
<query key=“machine">
   <target>cbd.resource.Machine<target>
   <caption>Query Machine</caption>
   <preCondition><![CDATA[id startsWidh
          “M”]]></preCondition>
   <filter>checkRelease</filter>
   <conditionAttributes>
       <attribute>id</attribute>
       <attribute>model</attribute>
   </conditionAttributes>
   <displayAttributes>
       <attribute>length</attribute>
       <attribute>critical</attribute>
   </displayAttributes>
</query>
```

Condition:

(ID startsWith MT) AND (length>200)

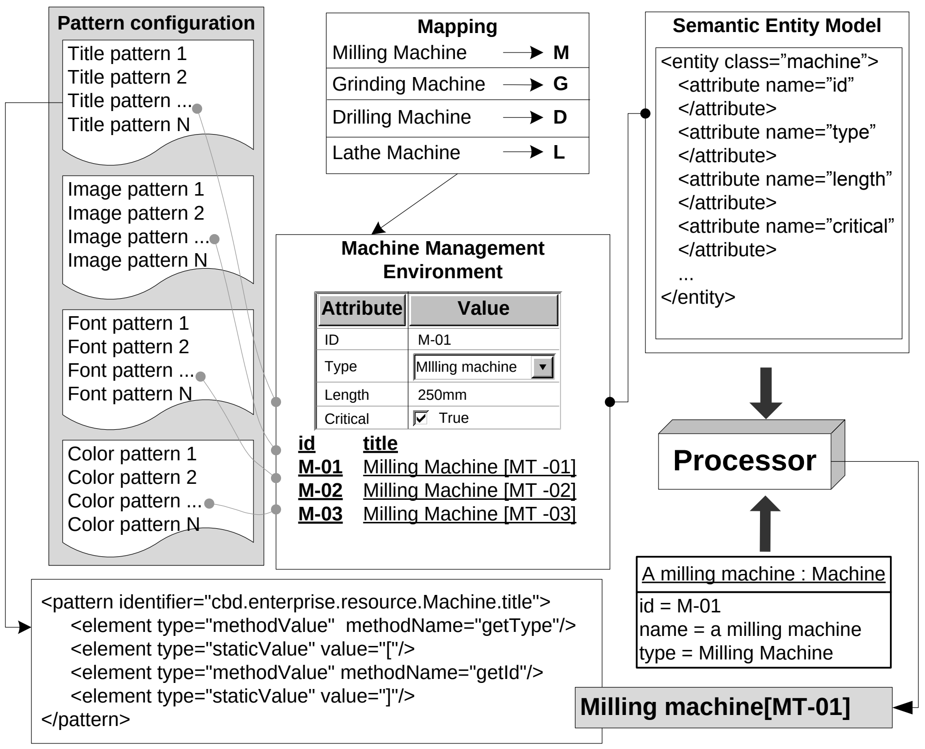
ID | starts with | 200

ID	Model	Length	Critical
MT-001	FUNAC-X5	250mm	☒ True
MT-002	SIMENS-45	380mm	☐ False
MT-003	AB-X214	700mm	☒ True
MT-004	MAHO-88	1000mm	☒ True
MT-005	DX-005	400mm	☐ False

Figure 19 Query component based on semantic representation

4.6 Semantic Graphical Environment Representation

Graphical environments are workplaces provided by enterprise systems for end users to view and manage information, or perform business activities. Flexible GUIs are critical to flexible enterprise systems. Entity management is selected to illustrate the semantic environment representation. Figure 20 shows an environment for managing machines. The environment adopts a two-column table as an attribute editing sheet. The editing sheet can provide different types of editors for each attribute based on a semantic entity model. Some specific editing components may be developed for some composite data types. In this case, editor can be defined in a semantic entity model. If an attribute has no editor configured, editors can be constructed according to attribute types. If a value source is defined for an attribute, a dropdown list may be created for the attribute. A check box can be created for a Boolean attribute. If the tag editable is a semantic attribute definition is set to false, the attribute is presented as a text.



```
<entity class="machine">
  <attribute name="id"
  </attribute>
  <attribute name="type"
  </attribute>
  <attribute name="length"
  </attribute>
  <attribute name="critical"
  </attribute>
  ...
</entity>
```

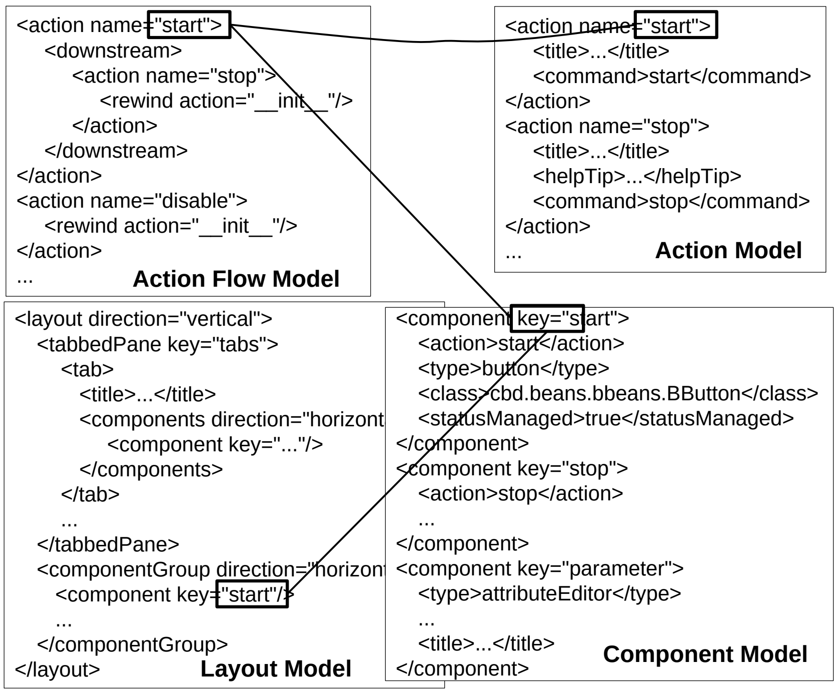
```
<pattern identifier="cbd.enterprise.resource.Machine.title">
    <element type="methodValue"  methodName="getType"/>
    <element type="staticValue" value="["/>
    <element type="methodValue" methodName="getId"/>
    <element type="staticValue" value="]"/>
</pattern>
```

Figure 20 Entity management environment

Various patterns can be configured to control the presentation of the machine list, such as title pattern, image pattern, and color pattern. As shown in Figure 20, a title pattern is provided for displaying the machine title, which is a combination of type and id. A title as "*Milling Machine [MT-01]*" can be derived for a given instance of machine based on the pattern. In this way, the presentation of the machine list is controlled by a set of patterns. When patterns are changed, the list can be presented in a different way. As mentioned above, some attribute values may be stored as keywords in a database but need to be presented in a more descriptive way. For example, in the database, "M", "L" and "D" represent milling machine, lathe machine and drilling machine respectively. Semantic mapping model can be used to support descriptive presentation. When a mapping model is associated with an attribute, a descriptive value is to be derived before presentation.

End users need to interact with a graphical environment to complete information management via various action components, such as buttons. A flexible entity management environment has to be able to present action components. A semantic scenario model is developed as shown in Figure 21. The semantic scenario model consists of four sub-models, which are the action model, action flow model, component model and layout model. The action model defines actions that end users can perform in a graphical environment. The component model defines various components that will be presented in a graphical environment. The layout model provides information about how action components are to be presented. In general, components are organized into different groups. A group can have multiple sub-groups. In a group, components can be arranged horizontally or vertically. Similarly, component groups at the same level can also be arranged horizontally or vertically. The layout model enables the definition of some specific containers for a group of components, such as tabbed or split panels. The action flow is for managing the state (active or inactive) of action components. The action flow model is used to evaluate the state of components at runtime. The state of component is to be re-evaluated after each user action. If a component needs a specific logic for managing its state, the value of the tag *statusManaged* in the component model should be set to "*false*". In this case, a specific state management bean can be configured for managing the state of the component.

It can be observed that semantic environment representation is achieved by the combination of semantic entity model, semantic environment model and semantic scenario model.



```
<action name="start">
    <downstream>
        <action name="stop">
            <rewind action="__init__"/>
        </action>
    </downstream>
</action>
<action name="disable">
    <rewind action="__init__"/>
</action>
...
```

```
<action name="start">
    <title>...</title>
    <command>start</command>
</action>
<action name="stop">
    <title>...</title>
    <helpTip>...</helpTip>
    <command>stop</command>
</action>
...
```

```
<layout direction="vertical">
  <tabbedPane key="tabs">
     <tab>
       <title>...</title>
       <components direction="horizont
          <component key="..."/>
       </components>
     </tab>
     ...
  </tabbedPane>
  <componentGroup direction="horizon
    <component key="start"/>
    ...
  </componentGroup>
</layout>
```

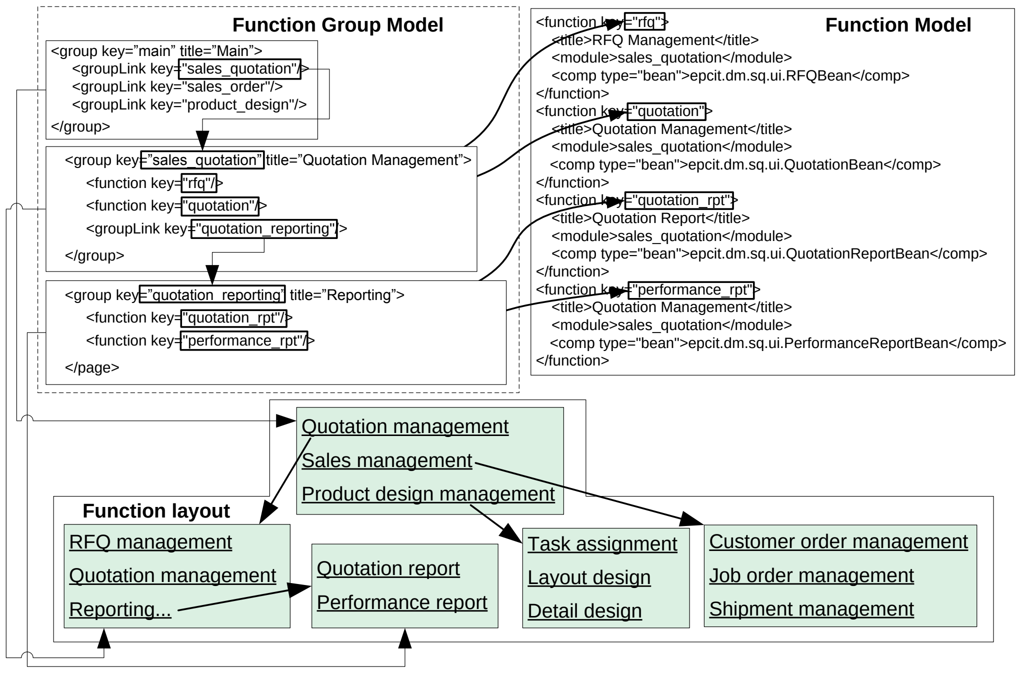
```
<component key="start">
   <action>start</action>
   <type>button</type>
   <class>cbd.beans.bbeans.BButton</class>
   <statusManaged>true</statusManaged>
</component>
<component key="stop">
   <action>stop</action>
   ...
</component>
<component key="parameter">
   <type>attributeEditor</type>
   ...
   <title>...</title>
</component>
```

Figure 21 Semantic scenario representation

4.7 Semantic Function Layout Representation

An enterprise system can be deemed as a collection of modules, such as a resource management module and a design management module. Each module may consist of various functions. Further, a function may be realized by a set of operations. For example, a quotation module may include functions like quotation management, quotation report management, and performance report management. In an enterprise system, system functions should be presented to end users in a logical way. A hierarchical structure is commonly adopted. The first level is initially presented to the user. By selecting an item, the user can invoke a concrete function or drill down to the next level. However, different companies may need different sets of functions and require different ways to group functions.

Compared to a simple application for individual activities, enterprise systems provide a large set of functions to support various activities performed by users from different departments. A flexible way to organize system functions is necessary. To achieve a flexible function layout, a semantic function layout model is developed. As shown in Figure 22, the function layout model consists of two sub-models: function model and function group model. The function model contains all functions to be presented to end users. In the function model, title, module and component are three key attributes that characterize system functions in terms of layout. A title is the descriptive name of a function. The module indicates a business domain that a function should be grouped into. The component specifies a functional component that actually realizes a function. The function group model is used to logically group functions for presentation. As shown in the Figure 22, various groups can be defined, and groups can be associated as a hierarchical structure to present as multi-level menus. In the *sales_quotation* group, the tag *groupLink* is used to link another group into this group. When the corresponding menu item is selected, a sub-menu is presented. The semantic function layout representation provides a unified method to group and present functions.



```
<group key="main" title="Main">
    <groupLink key="sales_quotation"/>
    <groupLink key="sales_order"/>
    <groupLink key="product_design"/>
</group>

  <group key="sales_quotation" title="Quotation Management">
      <function key="rfq"/>
      <function key="quotation"/>
      <groupLink key="quotation_reporting"/>
  </group>

  <group key="quotation_reporting" title="Reporting">
      <function key="quotation_rpt"/>
      <function key="performance_rpt"/>
  </page>
```

```
<function key="rfq">
   <title>RFQ Management</title>
   <module>sales_quotation</module>
   <comp type="bean">epcit.dm.sq.ui.RFQBean</comp>
</function>
<function key="quotation">
   <title>Quotation Management</title>
   <module>sales_quotation</module>
  <comp type="bean">epcit.dm.sq.ui.QuotationBean</comp>
</function>
<function key="quotation_rpt">
   <title>Quotation Report</title>
   <module>sales_quotation</module>
   <comp type="bean">epcit.dm.sq.ui.QuotationReportBean</comp>
</function>
<function key="performance_rpt">
   <title>Quotation Management</title>
   <module>sales_quotation</module>
  <comp type="bean">epcit.dm.sq.ui.PerformanceReportBean</comp>
</function>
```

Figure 22 Semantic function layout

4.8 Summary

This chapter presents and develops semantic representations for various types of system models, including entities, entity relationships, business logics and function layout. These semantic representations can be constructed by designers, developers or system administrators. In the meantime, they can also be effectively interpreted by computers. Therefore, they can be loosely coupled with software programs to control the behavior of software programs. Semantic representations make model driven enterprise systems highly configurable. Semantic models can be constructed based on the particular needs of a company during implementation. They can be reconfigured to support ongoing business changes after implementation.

Chapter 5 Industrial Case Identification

5.1 Introduction

Semantic representations developed in the previous chapter need to be further verified using industrial cases. This chapter develops a business process model based on the common practice of the manufacturing industry. Four critical business processes are identified. Based on the process model, resource management, product structure and reporting are identified as industrial cases for verifying semantic representations.

5.2 Business Processes in Manufacturing

Industrial cases need to be typical and representative. “Typical” implies that a case should have most characteristics concerned. “Representative” requires a case to have the need for flexibility. To identify proper industrial cases, it is necessary to understand business processes accurately. Due to the complexity of business processes, a graphical representation of business processes is necessary to provide an intuitive environment for effectively analyzing business processes. This research uses the ARIS Toolkit to document the business process model. ARIS is a well-known reference architecture, which mainly focuses on information systems for supporting business operation (Shin, et al. 2002). In ARIS, event driven process chain (EPC) diagrams provides an effective means to overview business processes in a hierarchical structure. These diagrams act as a control view to connect data view, function view and organization view. Figure 23 shows the first level of the business process model established based on the made-to-order practice of the manufacturing industry.

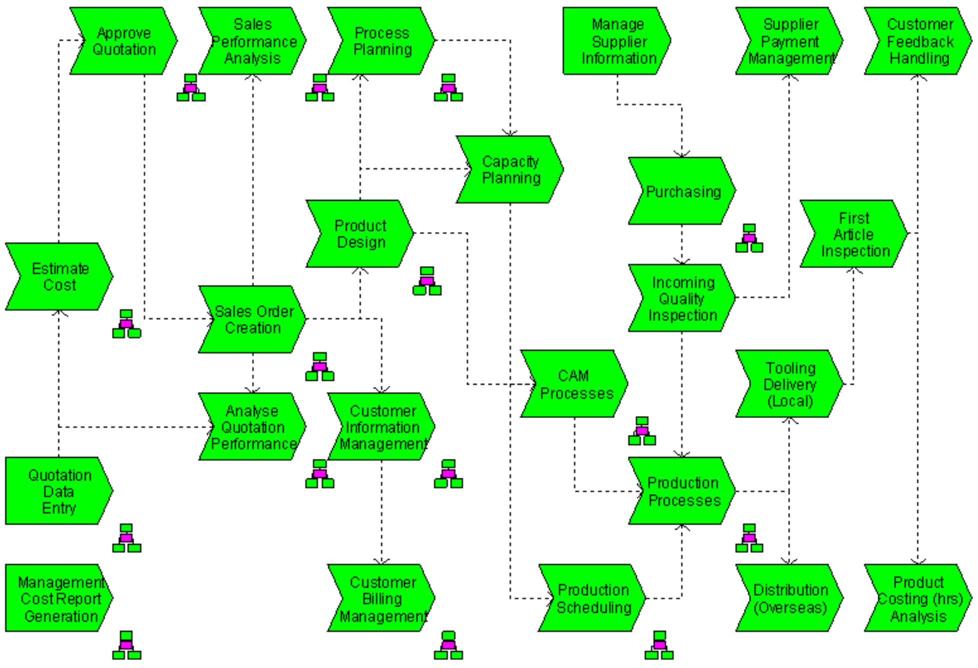

Approve Quotation
Sales Performance Analysis
Process Planning
Manage Supplier Information
Supplier Payment Management
Customer Feedback Handling
Capacity Planning
Purchasing
Product Design
First Article Inspection
Estimate Cost
Sales Order Creation
Incoming Quality Inspection
CAM Processes
Tooling Delivery (Local)
Analyse Quotation Performance
Customer Information Management
Quotation Data Entry
Production Processes
Management Cost Report Generation
Customer Billing Management
Production Scheduling
Distribution (Overseas)
Product Costing (hrs) Analysis

Figure 23 The first level EPC diagram.

5.3 Four Critical Business Processes

Through further analysis, four critical processes are identified as the order fulfillment process, design process, production process and material fulfillment process, as shown in Figure 24. The order fulfillment process is an interfacing process that connects manufacturing companies and customers. This process starts with the request for quotation (RFQ) and ends when the product is delivered; and the payment is made by customers. Major activities of this process include quotation preparation, customer order processing, project management and delivery management. When a quotation is requested, the sales department should promptly respond to the customer with a quotation report through requirement analysis and cost estimation. A quotation report usually encloses essential pricing information as well as terms and conditions. When a quotation is accepted, the customer sends a customer order to officially confirm the order. Long-term partners may directly send customer orders if trust relationships have been established. Typically, made-to-order manufacturing companies are operated in a project-centric manner. When a customer order is received, projects are initialized. After that, design, production and resources are managed on a project basis. After production, the customer service department takes the responsibility of organizing packing and delivery. At the same time, the finance department tracks and processes customer payments according to the payment terms described in the customer order.

The design process attempts to manage design activities and design information to improve design efficiency and quality. On receiving a customer order, the drawings of products and specific requirements are conveyed to the design department. First, layout design is carried out to determine the overall structure and the major specifications of key parts. After the layout design is confirmed internally, it may be sent to the customer for review and approval. Detailed design starts after the layout design is confirmed by the customer. Depending on the importance of parts, designers may be requested to submit their designs for review and approval. Finally, completed design is released for planning and production.

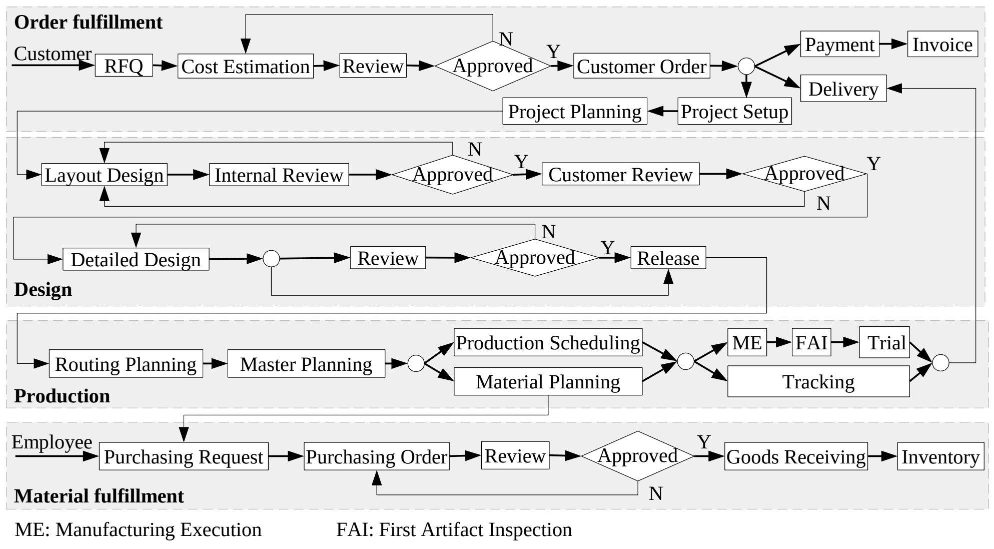


Figure 24 Critical processes in the mould making industry.

Production is a process to convert raw materials to products based on design specifications. It consists of two phases: planning and execution. In the planning phase, routing planning is first carried out to identify operations needed to machine each feature on individual parts, optimize the sequences of operations and features, and plan setups to position and clamp parts for machining. After routing planning, master planning is carried out to determine time periods during which individual parts must be accomplished so that the order due date can be fulfilled. Based on the outcome of master planning, production scheduling is further carried out to allocate time slots for each machining operation according to the available capacity of machines. To ensure that the required materials are ready for use when needed, material planning is also performed to determine the best time to issue material orders. After planning and scheduling, the production process runs to the execution phase. Workshop supervisors assign machining tasks to operators on a daily basis. The production progress is tracked and monitored according to the schedule.

The material fulfillment process manages purchase requests (PRs), purchase orders (POs) and inventory to support other business processes. According to material plans, the purchase department prepares and issues purchase orders to appropriate suppliers. Inventory management manages delivered materials and issues materials to workshops when requested. This process also manages non-planned consumer goods based on safety levels, which are the minimum quantities of individual goods to be maintained in inventory.

5.4 Integration Analysis

Four critical business processes are not independent. They need to be integrated to: 1) automate business activities; 2) motivate collaborative decision making; and 3) enable business process concurrency. In the following sections, the analysis of business process integration is conducted to identify commonly used information and key functions. The management of commonly shared information and key functions is to be selected as a case for further study.

5.5 Integrated Order Fulfillment Process

Resource information is needed in this process to determine if capacity and capability of available resources for early decision making can satisfy requirements of a customer order. If resources cannot satisfy a customer order, overtime and outsource may be required, which means high cost and long order lead time. In this case, a decision has to be made on the acceptance of the order with consideration given to customer satisfaction.

By leveraging the advantage of the similarity between quotations and customer orders, customer orders can be generated based on quotations and projects can be automatically initiated. When a project is initiated, relevant business units, such as design department, production department and purchasing department, can be notified with the necessary information. This greatly enhances the efficiency of customer order processing. Meanwhile, downstream teams start preparation and organize resources as early as possible. Consequently, the concurrency of business processes can be achieved.

Rapid responses to FRQ are critical in winning customer orders. An order fulfillment process needs to be integrated with other business processes for better cost estimation. On receiving a RFQ, the design department is to be notified with a request to perform product structure configuration. A product structure can be roughly estimated by identifying critical parts and materials for critical parts. During production structure configuration, process plans of critical parts can also be automatically created based on process plan templates. Process planners can be notified to review and refine process plans generated. With these types of information available, labor cost, equipment depreciation cost, material cost, packing cost and delivery cost are be considered in cost estimation to improve the accuracy of cost estimation.

5.5.1 Integrated Design

Design is a process of converting the needs and requirements for a product into specifications about a product. To fulfill customer requirements and allow the needs of downstream processes to be considered at an early stage, design process needs to be integrated with other critical processes.

Cost analysis is used to calculate the actual cost of customer orders after production. It is intended to improve the performance of cost estimation. By comparing reports of cost estimation and cost analysis, factors not considered accurately in cost estimation can be identified. Through the integration of design process with order fulfillment process, product structures configured in quotations and customer orders can be inherited to speed up the design process. As a result, seamless linkage can be achieved between the product structure configured in quotations or customer orders and the actual product structure for better cost analysis.

Integration of design process with production process enables manufacturability evaluation to be done at an early stage. It can effectively shorten the design cycle and minimize design iterations. This integration also enables partial releases in design for better process concurrency. Partial releases are done before design is completed and when design is mature for some downstream activities. A part can have multiple partial releases for different purposes, such as partial release for material purchasing and release for process planning.

For the purpose of better decision making in design, resource information needs to be made available to designers. This leads to better decision making in the selection of materials and material stocks for parts. Two ways exist for designers to select materials for parts. One is to specify a particular material stock available in inventory. Another is to simply define a blank if there is no appropriate material stock for use in inventory. In this case, a notification will be generated for the purchasing department to schedule purchase orders as early as possible. The information about cutting tools and machines also helps design to achieve better manufacturability.

5.5.2 Integrated production management

Process plans are documents containing detailed resource information and instructions for operators to complete operations. Process plan information is also the input of capacity planning and job scheduling. Process planners are responsible for creating process plans for each part to be machined internally. Workshop supervisors generate workshop tasks and job schedules based on product structures and process plans. These tasks are assigned to operators on a daily basis. The integration of production process with design process makes product structures available for determining the precedence of tasks. As mentioned above, concurrent engineering between design and production process can be achieved by partial release, process plans can be created as early as possible. Therefore, workshop tasks can be automatically generated at an early stage. This helps achieve better resource allocation, capacity management and workshop job scheduling. The production process can also be integrated with the order fulfillment process to make the progress of customer orders available to sales department and project management. This enables better customer satisfaction to be achieved through better communication with customers.

5.5.3 Integrated material fulfillment

Early purchase of materials not only holds cash in inventory but also increases the stock level of inventory. It introduces a barrier to the optimization of cash flow and leads to higher product cost. Late purchases can delay production, which can lead to the dissatisfaction in the order due date. This process also manages resource information which is widely shared by other business processes. It is imperative to integrate this process with design and production processes as analyzed above.

5.6 **Selection of Case Studies**

Throughout the above analysis, it can be observed that resource information and production structure are widely shared by critical business processes. As shown in Figure 25, resource information is widely shared by various business processes for different purposes. However, resource management capability is very limited if it is provided by traditional applications like CAD and CAM. Usually, traditional applications only manage some of manufacturing resources for internal use without attention to the needs of other applications. As most traditional applications are standalone and must be installed on individual computers, the coexistence of manufacturing resource information on different computers is inevitable. This research selects resource management as a case study by developing an extensible resource model based on semantic representations to support business process integration.

Product structure is also widely used in order fulfillment process, design process and production process. In the made-to-order environment, product structure is complicated because each product can have many variants with slightly different constitutions to fulfill different customer requirements. In such a context, product structure management needs two interrelated functions: family structure management and variant structure management. At the same time, these two functions need to be seamlessly integrated to ensure the consistency of a family structure and its variant structure. From the business process perspective, throughout the entire product lifecycle, different business activities look at the product structure for different purposes. Some activities are carried out based on variants and deem individual variants as different products and some activities need to be performed based on an entire family. A flexible product structure model is imperative. Therefore, product structure management for the made-to-order environment is selected as another case study for verifying the effectiveness of the proposed semantic representations.

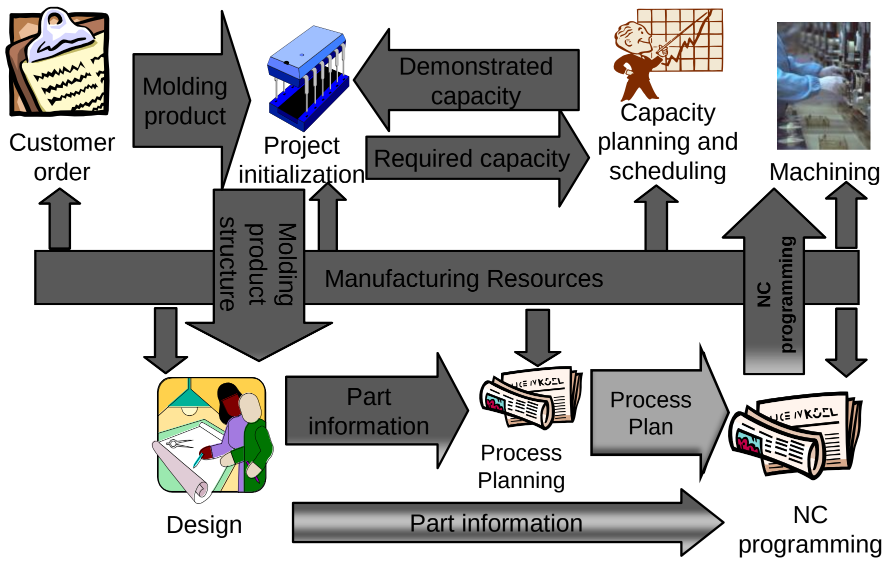


Figure 25 Resource information sharing

In addition, reports are critical deliverables provided to end users by enterprise systems. They provide structured and concise information for end users to effectively capture the status of resources, track the progress of jobs and analyze the profitability of products, etc. Reports are also key documents that help managers make decisions, perform planning activities and communicate with partners. At present, enterprise systems are usually developed with fundamental reporting capabilities. Much effort is required to design and develop customized reporting functions to individual companies at the implementation stage. Reporting customization is time-consuming and the result cannot be reused. Reports often have different formats and contents; Report look and feel varies from company to company. Therefore, reporting is a good study case for examining semantic representations.

5.7 Summary

The effectiveness of semantic representations needs to be further investigated using industrial cases. This chapter, in which four critical business processes are identified, develops a business process model based on the made-to-order practice in the manufacturing industry. By analyzing the requirements of integration between critical business processes, three case studies are selected, which are resource management, product structure management and reporting. The following three chapters a develop resource model, product structure model and reporting model, based on the three cases identified by using by semantic representations.

Chapter 6 Semantic Resource Modeling

6.1 Introduction

To effectively support business process integration, resource information needs to be managed in a centralized database for better sharing. It is imperative to look at resources from a broader viewpoint and represent resources in a flexible and unified way. A resource model is needed to coherently represent various resources so that resource information can be shared by each subsystem used by people with different concerns. Therefore, a comprehensive and extensible data structure needs to be developed to accommodate as much information as possible to characterize every aspect of resources and the relationships between their different types. By taking these needs into consideration, this chapter develops a semantic resource model as a case study of semantic representations.

6.2 Needs of Extensible Resource Model

Since the early 1990s, much effort has been put into research and development of information infrastructure and support platforms for business process integration (Aalst 2002). Most efforts were focused on reusing, extending and integrating various industry standards and enabling technologies, such as communications, object oriented technologies and also information exchange technologies, such as standards for the exchange of product model data (STEP) (Barry, et al. 1998, Camarinha-Matos and Afsarmanesh 1999b). Efforts can also be seen in the integration of multiple standards or technologies, e.g. exchange of business messages, exchange of technical product data, and federated/distributed database (Camarinha-Matos and Afsarmanesh 1999a).

Due to those efforts, IT infrastructure has evolved from being system-centric to network-centric, in order to facilitate information integration. It is moving towards a standard-centric and process-centric situation, as shown in Figure 26 (Liu 2003). The standard centric stage can be deemed as a leading stage to a process centric stage. In the standard centric infrastructure, standards play a very important role by providing interoperability among different heterogeneous systems, such as CORBA, STEP and ebXML. CORBA addresses interoperability among objects in different languages; STEP focuses on product information exchange between CAD applications and other applications. EbXML provides a standard means for business process modeling, which covers requirement analysis, architecture and collaboration (Lindsay, Downs and Lunn 2003).

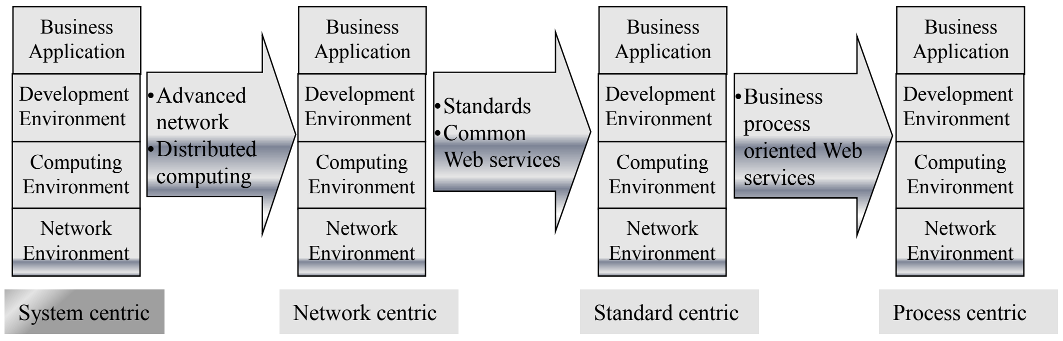


Figure 26 IT infrastructure trends (Liu, 2003)

Traditionally, resource models are functional based and created according to the needs of individual applications. Function-based models often lead to the problems of information inconsistency and duplication. They also hinder the integration of different applications due to the incompatibility between models. In order to better serve business process integration, resource management needs to be generalized as a core function, to support different business processes. On one hand, it needs to be generic enough to represent all resources. On the other hand, it should be developed in readiness to be extended for the representation and management of more details. Thus, it can be easily enhanced to support specific needs.

Extension of a resource model to manage specific information should not generate impacts to the entire model and existing resource management functions. Fundamental requirements of such a resource model include:

- Enabling incremental implementation to provide detailed resource information when needed;
- Enabling rapid implementation and deployment with less customization effort; and
- Providing a flexible foundation to support ongoing changes to business processes.

This case study develops a generic resource model, which can serve as a unified platform for process integration. The developed model consists of three interrelated sub-models: extensible object model, macro resource model and micro resource model. An extensible object model is an enabling technique based on semantic representations to make the resource model flexible and extensible. The macro model builds a generic framework for representing common characteristics of resources. The micro model enables specific characteristics to be represented.

6.3 **Foundation Model**

An enterprise system with the resource information hard-coded is rigid. It is difficult to tailor such as a system to support different business practices. To make an enterprise system generic, it is essential to develop a resource management engine which can be flexibly individualized for different needs. The resource model underneath such a resource management engine needs to be generic but also complete. By taking this into account, a unified resource model (shown in Figure 27) is developed. Figure 28 explains the symbols used in the model, which are the compliant of UML. To achieve generality and extensibility, the model characterizes resources from the three levels: foundation level, macro level and micro level.

The foundation level establishes an object-oriented model to support semantic representations. As shown in Figure 27, *AttributeDefinition* is modeled for representing and managing semantic attribute specifications in ADL. *IConfigurable* is the abstraction of entities that can have semantic entity models associated. *Configurable* is the default implementation of *IConfigurable* to provide capabilities shared by all subclasses. *AttributeValue* is for managing attribute values based on semantic attribute definition. *AttributeValues*, an aggregation of *AttributeValue*, is a collective data structure to manage the values of extended attributes based on semantic representations. *Configurable* aggregates *AttributeValues*. Fundamentally, this provides all its subclasses with the capability to support semantic representations and to hold the values of extended attributes.

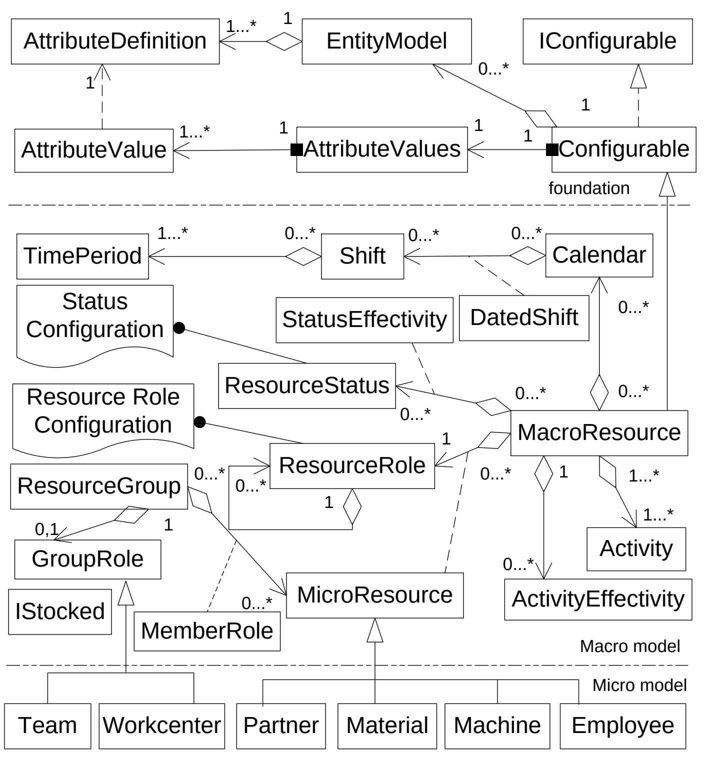


Figure 27 Extensible resource model

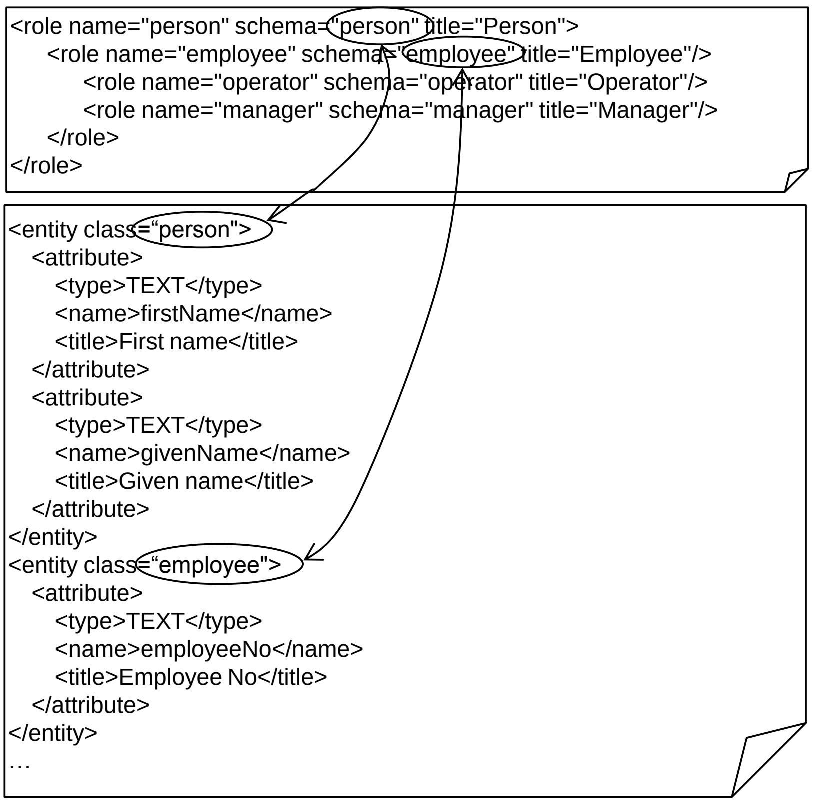
—▷	specializes
┈▷	implements
├┈	association class
◇—	aggregation by reference
■—	aggregation by value
→	reference association
┈>	dependency
—●	Configured by
A —c— B	c=1 means that one A instance has to be and only can have one B instance assoicated
	c=0,1 means that one A instance may have no or one B instance associated
	c=0...* means one A instance may have no or more than one B instances assicated
	No c means c=0,1

Figure 28 Legend explanations

6.4 Macro Resource Model

On the top of the extensible foundation, a macro model is established to characterize common features of resource entities. As shown in the model, *MacroResource* is a subclass of *Configurable*. *MacroResource* is the supper class of other resource entities. As a result, all resource entities can support semantic representations. In the macro model, *MacroResource* is a unified object modeled to represent all resources and *ResourceRole* plays the role of resource categorization. A resource can only have one role. *MicroResource* represents the association of *MacroResource* and *ResourceRole*. Thus, it also has a one-to-one mapping relationship with *MacroResource*. The information associated with *MacroResource* is the common information about resources. As shown in Figure 29, *ResourceRole* is defined in a semantic model. In the semantic model, the attribute *name* of the tag *role* is a unique key for identifying the role, and the attribute *title* defines the name of the role for screen display. The attribute *schema* specifies a key pointing to a semantic model associated with the role. As a result, each role can have a different set of attributes. *ResourceRole* is self-associated, which enables a role to have sub-roles. In other words, the resource can be categorized in the tree structure. A semantic model of a role can be inherited by its sub-roles and each sub-role also can have additional attributes to further characterize a resource associated with the role. As shown in the model, the resource with the employee role has three extended attributes. The attributes *firstName* and *givenName* are derived from the person role and the attribute *employeeNo* is specifically defined for the employee role. Semantic representations enable resource categorizations and attributes of each category to be configured or reconfigured. By changing resource role models, resource management engine can be tailored different enterprises.

The information about resource availability is critical to generate accurate plans and ensure that activities can be performed in time without resource conflicts. If the activities are not well planned, some activities may compete for resources. This can lead to the delay of the entire business process. The resource availability is described from two aspects, namely capacity and status.



```
<role name="person" schema="person" title="Person">
    <role name="employee" schema="employee" title="Employee"/>
        <role name="operator" schema="operator" title="Operator"/>
        <role name="manager" schema="manager" title="Manager"/>
    </role>
</role>
```

```
<entity class="person">
   <attribute>
      <type>TEXT</type>
      <name>firstName</name>
      <title>First name</title>
   </attribute>
   <attribute>
      <type>TEXT</type>
      <name>givenName</name>
      <title>Given name</title>
   </attribute>
</entity>
<entity class="employee">
   <attribute>
      <type>TEXT</type>
      <name>employeeNo</name>
      <title>Employee No</title>
   </attribute>
</entity>
…
```

Figure 29 Resource role configuration

6.4.1 Capacity Model

Resource capacity is information that is key to master planning and production scheduling. It is also utilized to manage task assignments in workshops and design departments. In the model, resource capacity is represented by a set of shifts and calendars. A shift consists of a set of time periods which indicates time slots when a resource takes effect. For instance, employees can be in the office from 8:00am to 5:00pm, but they are not available in the period from 12:00pm to 1:00pm, which is lunch time. Therefore, the shift for the employees consists of two time periods: 8:00am to 12:00pm and 1:00pm to 5:00pm. The calendar organizes shifts based on date on a yearly basis. If a date has more than one shift linked, it means that the related resource works for multiple shifts on that date.

6.4.2 Status Model

The capacity model describes the availability of resources over the timeline. However, at a certain time, a resource might not be available because of unexpected incidents, such as machine breakdown and staff leave. The status model assists to manage the availability of resources from this perspective. The status information is also used for performance analysis, cost tracking and task management. The model uses *ResourceStatus* to represent the resource status and *StatusEffectivity* to record the time period when a related resource stays in a particular status.

Different resources can have a different set of meaningful statuses. In addition, different companies may manage different sets of statuses for the same resource. Taking machines as an example, one company may only manage breakdown status in terms of availability, but another company may also manage monthly and yearly maintenance. To achieve flexibility in defining resource status, resource status is made configurable, as shown in Figure 30. The status configuration is associated with resources based on role. This implies that different roles can have different status.

```
<status role="machine">
    <identity>Normal</identity>
    <identity>Maintenance</identity>
    <identity>Breakdown</identity>
</status>
<status role="person">
    <identity>Normal</identity>
    <identity>Medical leave</identity>
</status>
```

Figure 30 Status configuration

6.5 Micro Model

MacroResource only holds information to all resources. Therefore, *MicroResource* is introduced to further characterize a specific resource. It can be seen from the model that specialization to *MicroResource* won't cause any impact to the macro level as *MicroResource* is the link class of *MacroResource* and *ResourceRole*. Therefore, the model is generic and extensible. This research focuses on three micro models: material model, machine model and cutting tool model.

6.5.1 Material Model

Material information is commonly shared by design, process planning and NC programming. Materials are characterized by physical properties and chemical properties. Designers should consider material physical properties, such as tensile strength, and select proper materials for parts to ensure the performance and life of parts and products. From the concurrent engineering viewpoint, material machining performance should also be considered in design. In processing planning, material chemical properties must be considered in the selection of cutting tools. In the machining process, high chemical affinity between a cutting tool material and a part material can cause sticking, even chemical changes, due to high temperature. This can dramatically decrease the quality of machined surfaces. Cutting tools made of tenacious materials are recommended to cut hard and brittle materials to minimize the chipping possibility of cutting edges. For tenacious part materials, cutting tools made of hard materials should be selected to achieve a high material removal rate by using high cutting speeds. In nature, cutting tool selection is a knowledge-intensive decision-making process. However, no sound theoretic model exists to represent this type of relationship, between cutting tool materials and part materials (Baker and Maropoulos 2000). To represent this knowledge, this research proposes two relationships between part materials and tool materials which are the compatible relationship and the repellent relationship. The former is the one which indicates that tool materials are highly recommended for associated part materials. The latter implies that tool materials are not appropriate for associated part materials.

Material information is shared by many activities in different processes, such as design, planning and purchasing. A unified material model, sharable to all related activities, can maximize material information sharing and streamline the relevant processes to improve working efficiency. Designers are responsible for selecting the proper materials for parts. In a design drawing, a designer can partially or fully specify material specifications, such as material code, raw status, shape and size. If only the material code is specified, a process planner has to decide on the blank shape and size by taking into consideration the required dimension accuracy, surface roughness and material machining performance. Material planning is done to decide when standard parts need to be ready for use and what material stocks are required. Material plans, outcomes of material planning, are official documents to support the purchase department in preparing and issuing purchase orders. In the market, shapes and sizes of material stocks are standardized. Non-standard material stocks can cost more and lead to longer supply time. Designers and planners look at materials in terms of part specification and performance. Purchasers are more concerned with prices, delivery time, shapes and sizes. At the micro level, the material model is specialized by incorporating the concerns discussed above, as shown in Figure 31.

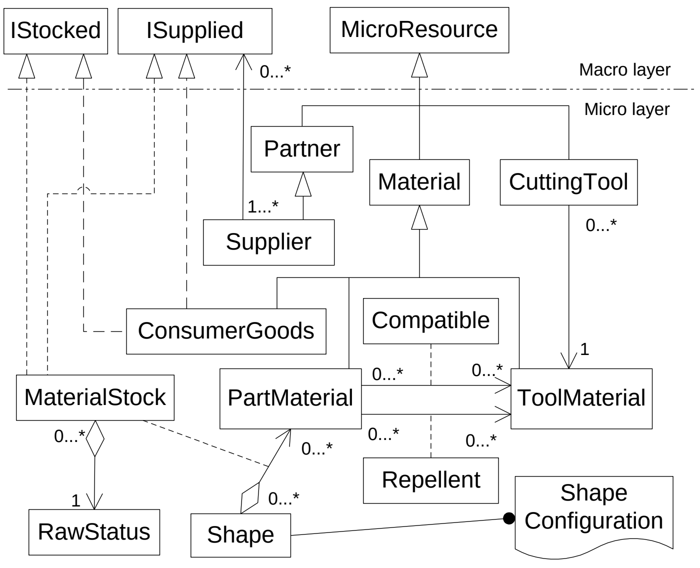


Figure 31 Micro material model

As shown in the model, *Material*, a subclass of *MicroResource*, is defined to represent materials. *PartMaterial* and *ToolMaterial* are specialized from *Material* to represent part materials and cutting tool materials respectively. *MaterialStock* is the association of *PartMaterial* and *Shape*, representing various sizes and shapes of the material stock. *PartMaterial*, an abstract concept without shape and size, provides information about chemical and physical properties. *MaterialStock* is used for characterizing shapes and sizes of materials. *Shape* can be configured using semantic representations based on the industrial standard. Standard sizes also can be configured in semantic models. *RawStatus* represents heat treatment to materials done by manufactures. As discussed above, *Compatible* and *Repellent* relationships are introduced and represent better and worse matches between cutting tool materials and part materials. This relationship can be represented using semantic models.

6.5.2 Micro Machine Model

Machines are one of the critical resources in manufacturing companies and machine information is needed by designers, planners, financial staff and maintenance staff. For the purpose of manufacturability evaluation, designers need to know the machine capability. The micro machine model is developed with focus on characterizing machine capability, as shown in Figure 32.

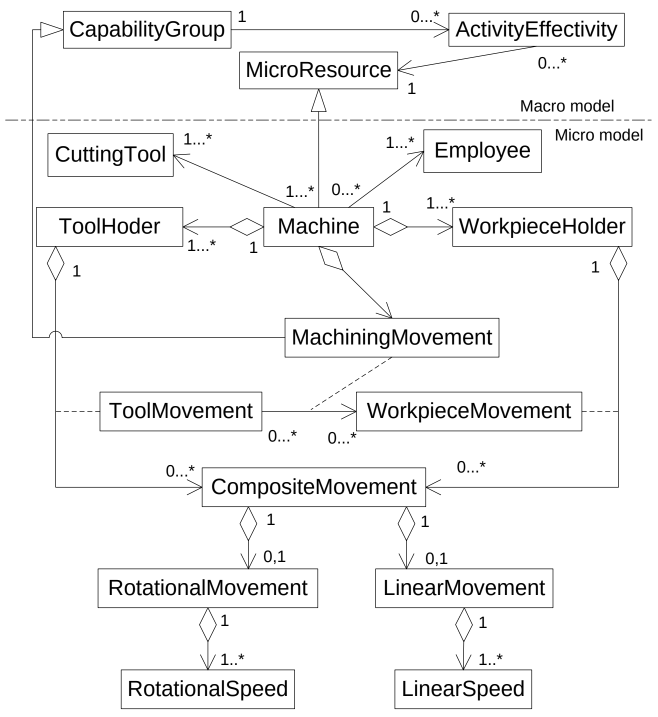


Figure 32 Micro machine model

Machining capability can be described by three elements: operations a machine can perform; design features a machine can deal with; and precision a machine can achieve. In general, one machine can carry out multiple types of operations. In addition, machine capabilities need to be associated with individual spindles or tool holders because each spindle or tool holder can perform different sets of operations. Theoretically, design features are formed by relative movements between cutting tools and work pieces. For instance, the combination of rotational movement of a work piece and linear movement of a cutting tool forms a cylindrical surface. Some machines, such as lathes, use the spindle as a work piece holder and others, such as milling machines, use the spindle as a tool holder. In order to make capability representation generic, abstract movements are used to assist defining machine capabilities. In the model, *LinearMovement* represents a linear movement characterized by *LinearSpeed*, which can be a set of values for discrete speed gearing or a speed range for continuous speed gearing. Similarly, *RotationalMovement* represents a rotational movement described by *RotationalSpeed* with a set of values or a speed range depending on the type of gearing. *CompositeMovement* represents composite movement of linear movement and rotational movement. As the model shows, *CompositeMovement* can have zero or one rotational movement and zero or one linear movement. Therefore, three types of *CompositeMovement* exist: 1) a composite movement with a *LinearMovement* linked, which means a linear movement; 2) a composite movement with a *RotationalMovement* linked, which means a rotation movement; and 3) a composite movement with both *LinearMovement* and *RotationalMovement*.

As shown in the model, a machine can have one or more tool holders and workpiece holders. Each tool holder and workpiece holder can link to zero or more *CompositeMovements*. Jointly, *ToolMovement* and *WorkpieceMovement* form a *MachiningMovement*. As shown in the model, *MachiningMovement* is specialized from *CapabilityGroup*. As a result, each *MachiningMovement* instance can have a set of capabilities.

6.5.3 Micro cutting model

Effective cutting tool information available during the design stage can assist designers to make better decisions to prevent the proliferation of cutting tools, which is the problem that nearly 70% of manufacturing companies are struggling to tackle (Baker, et al. 2000). This can lead to less manufacturing lead time and lower production cost. The micro cutting tool model is developed, as shown in Figure 33. The model can represent the complicated tools with multiple cutting edges. For the purpose of manufacturability evaluation, three major types of information, i.e. tool shank, tool material and cutting edges, are represented. Tool shank, an interface to the tool holder of machines, is represented by *Shank*, a subclass of *Shape* defined at the macro level. Similar to machine tools, cutting tool capabilities are represented as the combination of operation, feature and precision. Cutting tool capabilities are managed based on the cutting edge, represented by *CuttingEdge*.

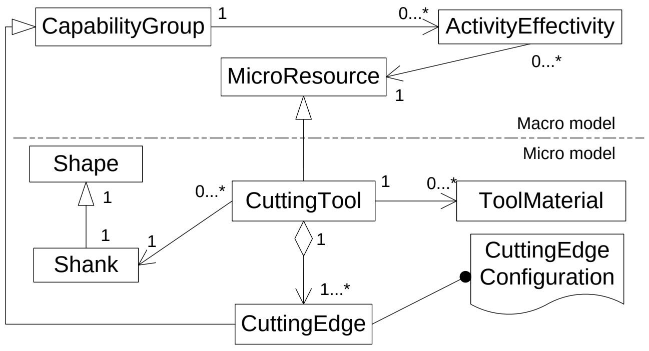
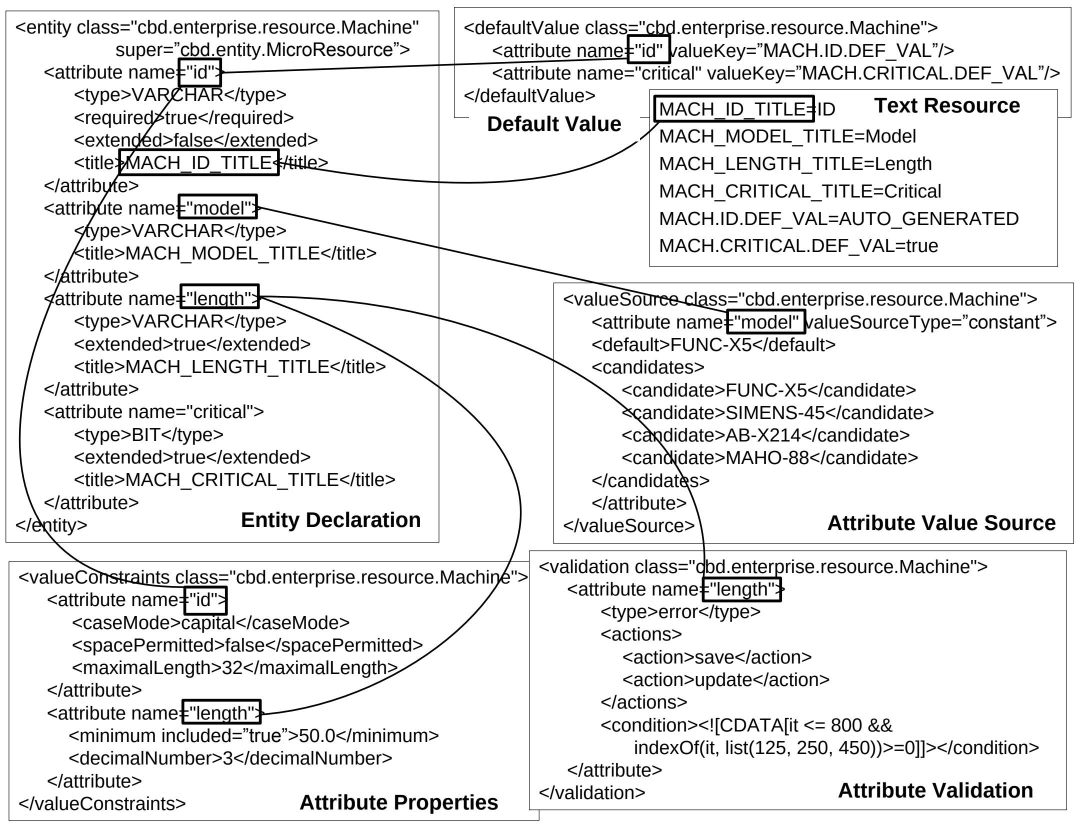


Figure 33 Micro cutting tool model

6.6 Illustration of Semantic Resource Representations

This section briefly demonstrates the semantic representation for machines. Based on semantic entity representations, the semantic machine model can consist of six elements, as shown in Figure 34:

1) Entity declaration, which defines attributes the machine has. Machine entity is defined on the top of a super class named *cbd.entity.MicroResource*. It inherits all attributes defined for *cbd.entity.MicroResource*;
2) Value constraints, which provide simple rules for validating attribute values;
3) Value validations, which provide comprehensive rules for validating attribute values based on business rules. This often involves validating relationships with other entities;
4) Default values, which define default values of machine attributes
5) Text resource, which provides a way to achieve the localization of default values and attribute titles.
6) Value sources, which provide a way to get value candidates of attributes. As shown in the model, machine model should be selected from a list.



```
<entity class="cbd.enterprise.resource.Machine"
            super="cbd.entity.MicroResource">
    <attribute name="id">
        <type>VARCHAR</type>
        <required>true</required>
        <extended>false</extended>
        <title>MACH_ID_TITLE</title>
    </attribute>
    <attribute name="model">
        <type>VARCHAR</type>
        <title>MACH_MODEL_TITLE</title>
    </attribute>
    <attribute name="length">
        <type>VARCHAR</type>
        <extended>true</extended>
        <title>MACH_LENGTH_TITLE</title>
    </attribute>
    <attribute name="critical">
        <type>BIT</type>
        <extended>true</extended>
        <title>MACH_CRITICAL_TITLE</title>
    </attribute>
</entity>
```

```
<defaultValue class="cbd.enterprise.resource.Machine">
    <attribute name="id" valueKey="MACH.ID.DEF_VAL"/>
    <attribute name="critical" valueKey="MACH.CRITICAL.DEF_VAL"/>
</defaultValue>
```

```
MACH_ID_TITLE=ID
MACH_MODEL_TITLE=Model
MACH_LENGTH_TITLE=Length
MACH_CRITICAL_TITLE=Critical
MACH.ID.DEF_VAL=AUTO_GENERATED
MACH.CRITICAL.DEF_VAL=true
```

```
<valueSource class="cbd.enterprise.resource.Machine">
    <attribute name="model" valueSourceType="constant">
    <default>FUNC-X5</default>
    <candidates>
        <candidate>FUNC-X5</candidate>
        <candidate>SIMENS-45</candidate>
        <candidate>AB-X214</candidate>
        <candidate>MAHO-88</candidate>
    </candidates>
    </attribute>
</valueSource>
```

```
<valueConstraints class="cbd.enterprise.resource.Machine">
    <attribute name="id">
        <caseMode>capital</caseMode>
        <spacePermitted>false</spacePermitted>
        <maximalLength>32</maximalLength>
    </attribute>
    <attribute name="length">
       <minimum included="true">50.0</minimum>
       <decimalNumber>3</decimalNumber>
    </attribute>
</valueConstraints>
```

```
<validation class="cbd.enterprise.resource.Machine">
    <attribute name="length">
        <type>error</type>
        <actions>
            <action>save</action>
            <action>update</action>
        </actions>
        <condition><![CDATA[it <= 800 &&
            indexOf(it, list(125, 250, 450))>=0]]></condition>
    </attribute>
</validation>
```

Figure 34 Semantic machine representation

Figure 35 shows views defined for the machine. A view contains a subset of entity attributes for different roles. Views assist access control and tailor information for different business units. The view model indicates the view 1 is editable and view 2 is read-only. If a role is associated with view 2, all persons with this role can only view machine id and machine name. A machine can have a supervisor and multiple operators associated. These are reflected by relationship models shown in Figure 36. The relationship model implies that a machine can only have one supervisor but one supervisor can be associated with multiple machines. The relationship between machines and operators are many-to-many. According to this relationship definition, the machine-supervisor relationship is exclusive, which means that when a person is associated with a machine as supervisor, this person cannot be associated with the machine in other relationships. Vice versa, if a person already has other relationships with a machine, he cannot be associated with the machine as a supervisor.

```
<object class="cbd.enterprise.resource.Machine">
        <view name="cbd.enterprise.resource.Machine.view1">
                <attribute name="id">
                     <editable>false</editable>
                </attribute>
                 <attribute name="name">
                     <editable>true</editable>
                </attribute>
                <attribute name="worktableLength">
                     <editable>true</editable>
                </attribute>
         </view>
         <view name="cbd.enterprise.resource.Machine.view2" >
                 <attribute name="id">
                      <editable>false</editable>
                </attribute>
                 <attribute name="name">
                      <editable>false</editable>
                </attribute>
         </view>
 </object>
```

Figure 35　Machine view

```
<relationship key="machine-person" roleA="cbd.enterprise.resource.Machine"
                                                          roleB="cbd.enterprise.org.Person">
     <schema key="supervisor">
             <title>_localizedText(RELATION_SUPERVISOR)</title>
             <linkRole>supervisor</linkRole>
             <linkClass>cbd.enterprise.resource.MachinePersonLink</linkClass>
             <roleA>
                     <primaryKey>
                             <attribute name="id" nameInLink="machineId"/>
                     </primaryKey>
                     <cardinality>M</cardinality>
             </roleA>
             <roleB>
                     <primaryKey>
                             <attribute name="id" nameInLink="personId"/>
                     </primaryKey>
                     <cardinality>1</cardinality>
                     <exclusive>true</exclusive>
             </roleB>
     </schema>
     <schema key="operator">
             <title>_localizedText(RELATION_OPERATOR)</title>
             <linkRole>operator</linkRole>
             <linkClass>cbd.enterprise.resource.MachinePersonLink</linkClass>
             <roleA>
                     <primaryKey>
                             <attribute name="id" nameInLink="machineId"/>
                     </primaryKey>
                     <cardinality>M</cardinality>
             </roleA>
             <roleB>
                     <primaryKey>
                             <attribute name="id" nameInLink="persionId"/>
                     </primaryKey>
                     <cardinality>M</cardinality>
             </roleB>
     </schema>
</relationship>
```

Figure 36　Relationships with machine

6.7 Summary

By considering the requirements of process integration, the resource model is separated into three levels: foundation, macro and micro. The foundation layer is a linkage between the resource model and semantic representations. At the macro level, the model is generic to represent all resources by addressing role, availability, capability and group. At the micro level, the model can be extended to present further detailed information of a specific resource. With various semantic representations incorporated, the model is also highly configurable. This case study proves that semantic representations can be effectively applied together with traditional modeling methods to establish configurable and extensible business object models.

Chapter 7 Semantic Product Structure Modeling

7.1 Introduction

Product structure is a hierarchical tree representing the classification of components of a product and the interrelationships of components. Product structure is key information widely shared by various business activities performed at different stages (Mannisto, Peltonen, Martio and Sulonen 1998, Eynard, Gallet, Nowak and Roucoules 2004, He, Ni, Ming and Lu 2004). PLM is a strategic business principle to make product information consistent and sharable throughout the entire product lifecycle. It associates relevant information with product structures throughout business processes to serve various needs (Thimm, Lee and Ma 2006). This case study first develops a comprehensive product structure based on the needs of the made–to-order manufacturing environment. Then, semantic representations are applied to represent entities and categorizations in product structure to verify the effectiveness of semantic representations.

7.2 Background

To provide customers with tailor-made products faster, better and cheaper, manufacturers have shifted their production mode to mass customization in order to take advantage of mass production for small batch-size production (Ni, Ming and Lu 2003). For such an environment, a product initially consists of a common base and modularized functional subsystems to form a customization platform (MacCarthy, Brabazon and Bramham 2003, Brière-Côté, Rivest and Desrochers 2010). Accordingly, it is essential to develop a product structure model capable of flexibly representing product families and product variants with the attention to different business processes in a product lifecycle (Xu and Jiao 2009a). A good product structure model should be able to synchronize a family structure and its variant structures (Xu and Jiao 2009b). At present, research in this area generally attempts to structure and represent detailed data related to a single product, and many product structure models and associated management systems are specifically developed for the purpose of design management. Product structure models consider product family and are capable of supporting entire product lifecycle management rarely exists (Shu and Wang 2005).

One essential function of PDM systems is to manage product structure (Eynard, et al. 2004). However, few available PDM systems are powerful enough to effectively manage product structures for mass customization because of the weakness of product structure models in representing product families (Janitza, Lacher, Maurer, Pulm and Rudolf 2003). Additionally, as the current PDM framework is specifically defined for design management, most product structure models underlying PDM systems lack the ability to support the integration of other business processes, such as customer order management, planning and production (Hameri and Nihtila 1998, He, Ni and Lee 2003).

Reports can be found on product structure models relevant to product family representation. Sudarsan (Sudarsan, Fenves, Sriram and Wang 2005) presented a product information modeling framework based on three models: open assembly model (OAM), design-analysis integration model (DAIM) and product family evolution model (PFEM). Of these models, PFEM model addresses product family representation. It pays little attention to the effective representation of common characteristics of a family and particular characteristics of a variant. Du (Du, Jiao and Tseng 2000) reported a product structure model to represent product family for mass customization. The model employs three views, which are functional view, technical view and structural view. The functional view focuses on the classification of diverse functional features of a product portfolio for customer recognition. The technical view is intended to represent building blocks. The structural view represents the topological structure of building blocks and configuration rules guide the product configuration. This model is helpful for companies in shifting from individual product development to family-based design because it provides a systematic method to establish a building block repository and configuration rules. However, it lacks the ability to support design process management. Fujita (Fujita 2002) proposed a product structure representation by decomposing a product into different subsystems. By employing entity relationships to represent the topological structure of subsystems and attributes to represent the association possibilities of subsystems, the model places its focus on maximizing product varieties using minimum building blocks to achieve optimized a customization platform. Janitza (Janitza, et al. 2003) also reported a product model for mass customization by incorporating product decomposition and part specification into one model. This model focuses on providing a flexible product model specification for product designers and a simple configuration for customers. Family representation and variant representation has not received enough attention, and little research can found on the synchronization of family presentation and variant representation.

This case study addresses some of main gaps, which include: 1) explicit representations of common characteristics of product family and specific features of product variants; 2) synchronization of a family model and its variant models in the context of mass customization; 3) integration of production structure and other business object models; and 4) extensibility of product structure models for flexible product lifecycle management systems.

7.3 **Abstract Product Structure Model**

7.3.1 Master-variant pattern

To enable the developed model to effectively represent the common features of a family and special features of different variants, a master-variant pattern, as shown in Figure 37, is proposed as a fundamental technique for establishing the product structure model. In the model, the interfaces *IMaster* and *IVariant* are modeled to represent common properties and behaviors of families and variants respectively. The interface *IMVLink* represents common properties and behaviors of associations between masters and variants. Classes that directly or indirectly implement the interface *IMaster* are enforced to comply with the principles defined by the master-variant pattern. The cardinalities of the association between *IMaster* and *IVariant* imply that one master can have one or more variants and a variant should have and can only have one master. Based on this pattern, a master and its variants exist interdependently. A master cannot exist without a variant and vice versa. Attributes common to all variants should be defined in the master classes, which are the classes that directly or indirectly implement the interface *IMaster*. Attributes specific to variants should be modeled in variant classes, which directly or indirectly implement the interface *IVariant*. *IMaster* is an abstract for grouping variants and represents the common characteristics of variants. *IVariant* represents the special characteristics of variants.

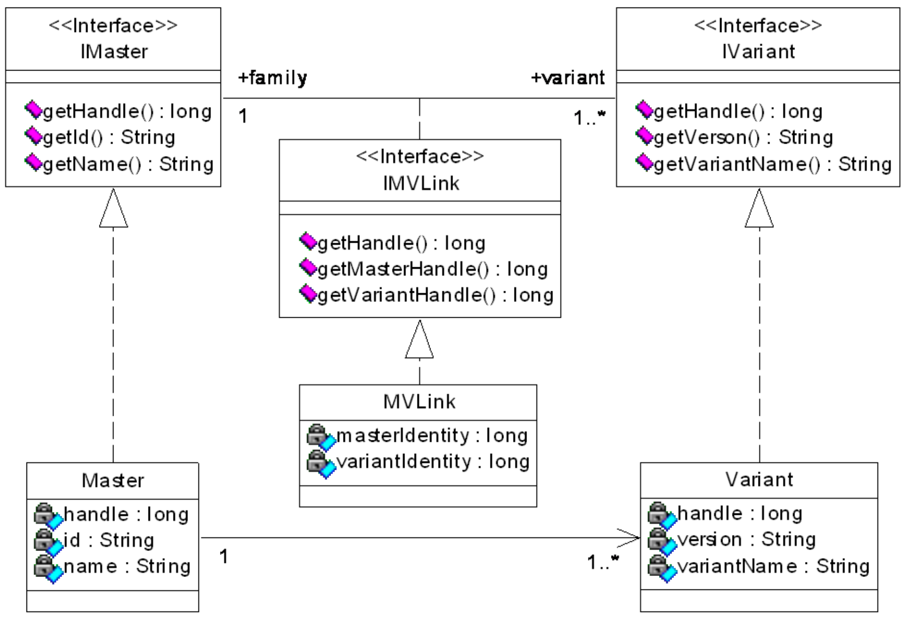


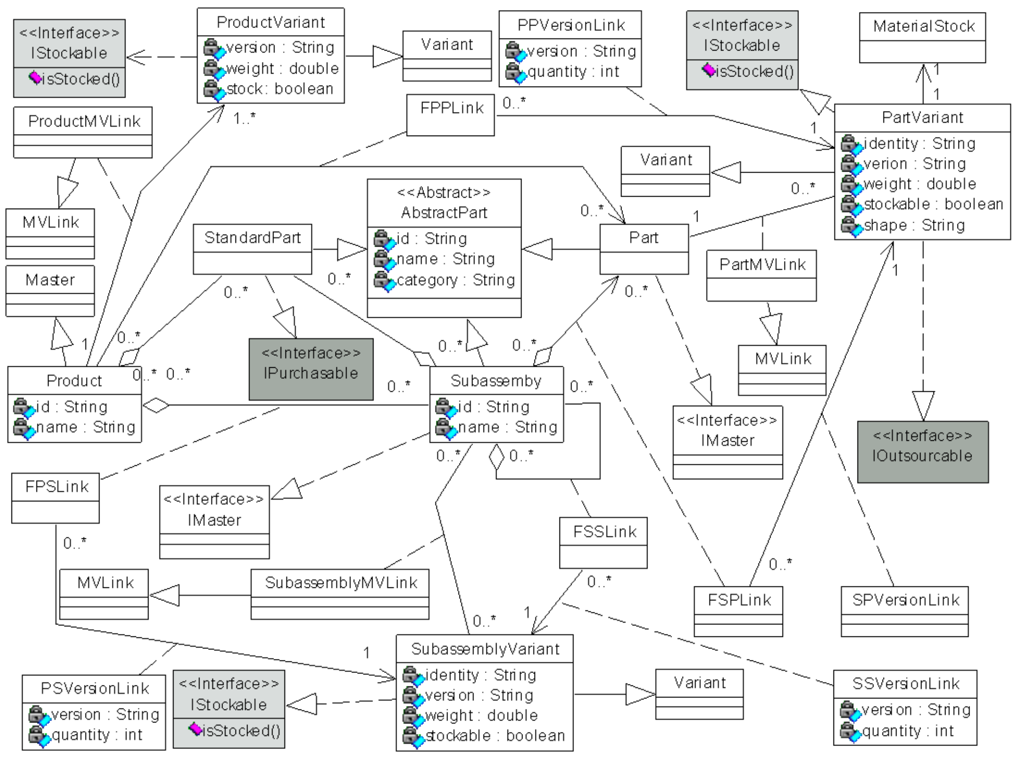
Figure 37 Master-variant pattern.

In the model, the attributes *id* and *name* are defined to uniquely identify individual families. The attribute *version* is used to differentiate variants in a family. The model implies that all variants share the same id and name and each variant can have a special name because the attribute *variantName* is defined in the class *Variant*.

The master-variant pattern offers three main advantages: 1) it provides a clear boundary between family representation and variant representation. At the same time, it provides an easy way to maintain data integrity; 2) it is capable of representing common characteristics of families and specific characteristics of individual variants; and 3) it can flexibly meet different requirements of different business processes. Masters or variants can be explicitly used as input to business processes and related information can be explicitly linked to masters or variants. For example, process plans can be linked to variants so that each variant may be produced in a different set of operations. Assembly plans may be linked to masters as all variants have the same connection features.

7.3.2 Product structure model

Based on the master-variant pattern, the product structure model shown in Figure 38 is developed. Based on the master-variant pattern, the model adopts three groups of classes to represent product, part and subassembly respectively: *Product*, *ProductVariant* and *ProductMVLink*, *Part*, *PartVariant* and *PartMVLink* as well as *Subassembly*, *SubassemblyVariant* and *SubassemblyMVLink*. The classes *Product*, *Part* and *Subassembly* represent product masters, part master and subassembly master while the classes *ProductVariant*, *PartVariant* and *SubassemblyVariant* represent product variants, part variants and subassembly variants.



<<Interface>> IStockable isStocked()
ProductVariant version : String weight : double stock: boolean
Variant
PPVersionLink version : String quantity : int
<<Interface>> IStockable isStocked()
MaterialStock
ProductMVLink
FPPLink
PartVariant identity : String verion : String weight : double stockable : boolean shape : String
Variant
MVLink
StandardPart
<<Abstract>> AbstractPart id : String name : String category : String
Part
PartMVLink
Master
<<Interface>> IPurchasable
MVLink
Product id : String name : String
Subassemby id : String name : String
<<Interface>> IMaster
<<Interface>> IOutsourcable
FPSLink
<<Interface>> IMaster
FSSLink
MVLink
SubassemblyMVLink
FSPLink
SPVersionLink
PSVersionLink version : String quantity : int
<<Interface>> IStockable isStocked()
SubassemblyVariant identity : String version : String weight : double stockable : boolean
Variant
SSVersionLink version : String quantity : int
1..*
0..*
1

Figure 38 Product structure model.

7.3.2.1 Family Structure

For clarity, the family structure model is taken out from Figure 38 and shown in Figure 39. In the model, aggregation associations between *Product* and *Part*, *Subassembly* as well as *StandardPart* indicate that a product can consist of non-standard parts, subassemblies and standard parts. A subassembly can constitute other subassemblies, non-standard parts and standard parts. Therefore, *Subassembly* has aggregation associations with itself, *Part* and *StandardPart*. It has to be pointed out that *Part* and *Subassembly* are master classes, as shown in Figure 39, they represent a family rather than a concrete variant. Hence, the model shown in Figure 39 only reflects how families, subassembly families and standard parts are involved in a product family. It does not provide concrete information about which variant of a part family or a subassembly family is involved in a product variant. However, based on the master-variant link, all part variants and subassembly variants are clearly reflected. Therefore, the family model provides an overall view of a product family about product variants and all optional part variants and subassembly variants. Such an overview is called product family spectrum (Hameri, et al. 1998).

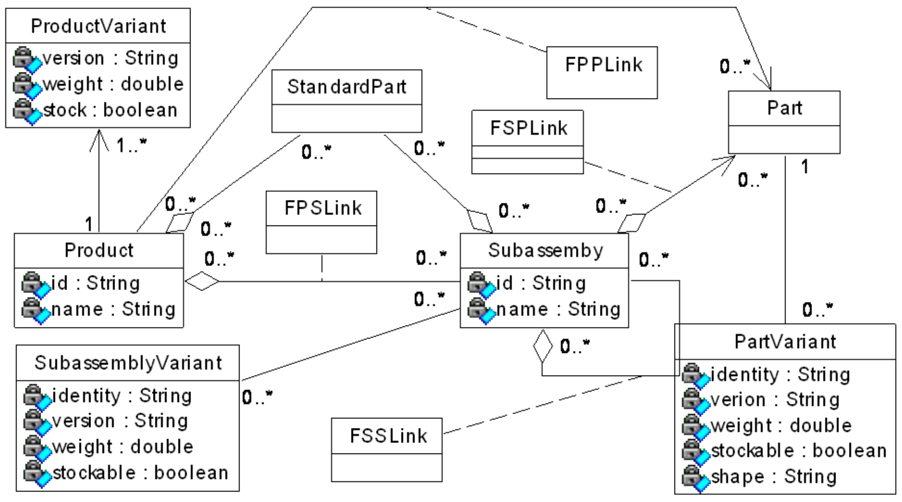


Figure 39 Product family model.

Figure 40 shows the sample spectrum view of a simplified car family based on the developed model. A car family, represented by *Car*:*Product*, can consist of an audio subsystem, represented by *Audio*:*Subassembly*, and an engine, represented by

Engine:*Part*. Further, an audio subassembly consists of a radio subsystem, represented by *Radio*:*StandardPart*, and a media player, represented by *MediaPlayer*:*Subassembly*. From the spectrum, it can be clearly seen that three types of engines with different rated powers and three types of audio subsystems, which are cassette player, CD player and video player, are available for selection.

The spectrum can effectively assist designers to configure products for customers, amend design to reorganize existing functions into configurable subsystems, design new alternative subsystems, or develop new functional subsystems to enhance customizability of a family. It also helps customers configure products when placing orders.

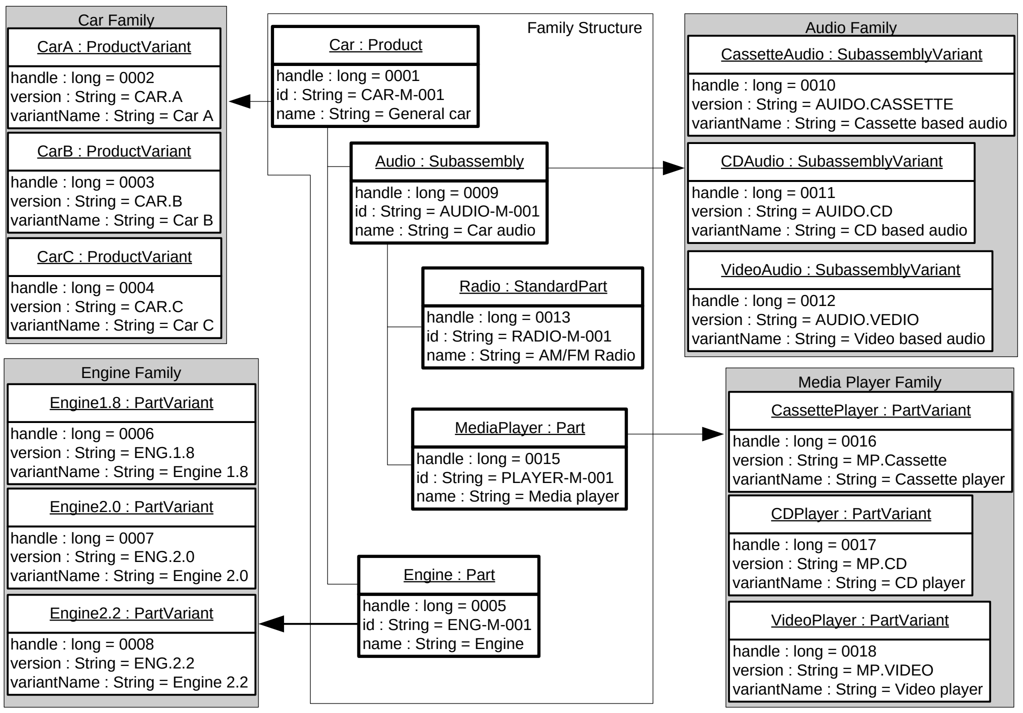

Car Family
CarA : ProductVariant
handle : long = 0002
version : String = CAR.A
variantName : String = Car A
CarB : ProductVariant
handle : long = 0003
version : String = CAR.B
variantName : String = Car B
CarC : ProductVariant
handle : long = 0004
version : String = CAR.C
variantName : String = Car C
Family Structure
Car : Product
handle : long = 0001
id : String = CAR-M-001
name : String = General car
Audio : Subassembly
handle : long = 0009
id : String = AUDIO-M-001
name : String = Car audio
Radio : StandardPart
handle : long = 0013
id : String = RADIO-M-001
name : String = AM/FM Radio
MediaPlayer : Part
handle : long = 0015
id : String = PLAYER-M-001
name : String = Media player
Engine : Part
handle : long = 0005
id : String = ENG-M-001
name : String = Engine
Audio Family
CassetteAudio : SubassemblyVariant
handle : long = 0010
version : String = AUIDO.CASSETTE
variantName : String = Cassette based audio
CDAudio : SubassemblyVariant
handle : long = 0011
version : String = AUIDO.CD
variantName : String = CD based audio
VideoAudio : SubassemblyVariant
handle : long = 0012
version : String = AUDIO.VEDIO
variantName : String = Video based audio
Engine Family
Engine1.8 : PartVariant
handle : long = 0006
version : String = ENG.1.8
variantName : String = Engine 1.8
Engine2.0 : PartVariant
handle : long = 0007
version : String = ENG.2.0
variantName : String = Engine 2.0
Engine2.2 : PartVariant
handle : long = 0008
version : String = ENG.2.2
variantName : String = Engine 2.2
Media Player Family
CassettePlayer : PartVariant
handle : long = 0016
version : String = MP.Cassette
variantName : String = Cassette player
CDPlayer : PartVariant
handle : long = 0017
version : String = MP.CD
variantName : String = CD player
VideoPlayer : PartVariant
handle : long = 0018
version : String = MP.VIDEO
variantName : String = Video player

Figure 40 A simplified car family spectrum.

7.3.2.2 *Variant Structure*

A variant structure should clearly reflect what part variants and subassembly variants are used to form a particular product variant. At the same time, the model should be capable of enforcing the consistency of the family structure and variant structures. To achieve this goal, the variant structure model is built on the top of the family structure model. As shown in Figure 38, *FPPLink* and *FPSLink* respectively represent associations of a product family with a part family and a subassembly family, and *FSSLink* represents association of a subassembly family with other subassembly families. To further represent variant structures, three association classes, i.e. *PPVersionLink*, *PSVersionLink* and *SSVersionLink*, are defined to associate *FPPLink* with *PartVaraint*, *FPSLink* with *SubassemblyVariant* and *FSSLink* with *SubassemblyVariant*. *PPVersionLink*, *PSVersionLink* and *SSVersionLink* are called version links. A key attribute in the version links is version. The value of this attribute indicates to which product variant or subassembly variant the associated variant is attached.

To explain the variant structure model, the relationships between a car variant and engine variants are taken as an example. As shown in Figure 41, the car family has three variants, i.e. *CarA*, *CarB* and *CarC*, and the engine family also has three variants, which are *Engine1.8*, *Engine2.0* and *Engine2.2*. *Car* and *Engine* are associated through *CarEngineLink*, which is an instance of *FPPLink*. As mentioned above, *FPPLink* is incapable of providing information about which engine variant is used for *CarA*, *CarB* and *CarC* respectively. To reflect the associations between the engine variants and the car variants, three version link instances are introduced, i.e. *EngineVersionLink1*, *EngineVersionLink2* and *EngineVersionLink3* to associate *Engine1.8*, *Engine2.0* and *Engine2.2* with *CarEngineLink* respectively. The attribute *version* in the version link classes plays the role of specifying which car variant each associated engine variant is used for. According to Figure 41, it is clear that *Engine1.8* is used for *CarA,* as the value of the attribute version of *EngineVersionLink1* is *CAR.A*, which should be same as that of the attribute version of *CarA*.

Compared to the variant structure model that directly associates variants of a part family and subassembly family with a product variant, a significant advantage of this model is that the family structure model and the variant structure model are integrated. As a result, product variant structures can be well controlled by the corresponding product family structure. For example, in Figure 41, if the engine family was not associated with the car family, *CarEngineLink* would not exist. Consequently, no engine variants could be associated with any product variants.

This feature is very significant to companies which manage multiple families and there exist multiple subsystems that provide the same functions, but are not exchangeable crossover families. For instance, two engine families are maintained for two car families respectively without exchangeability. While configuring products, this model can effectively prevent the selection of incompatible variants based on the developed product structure.

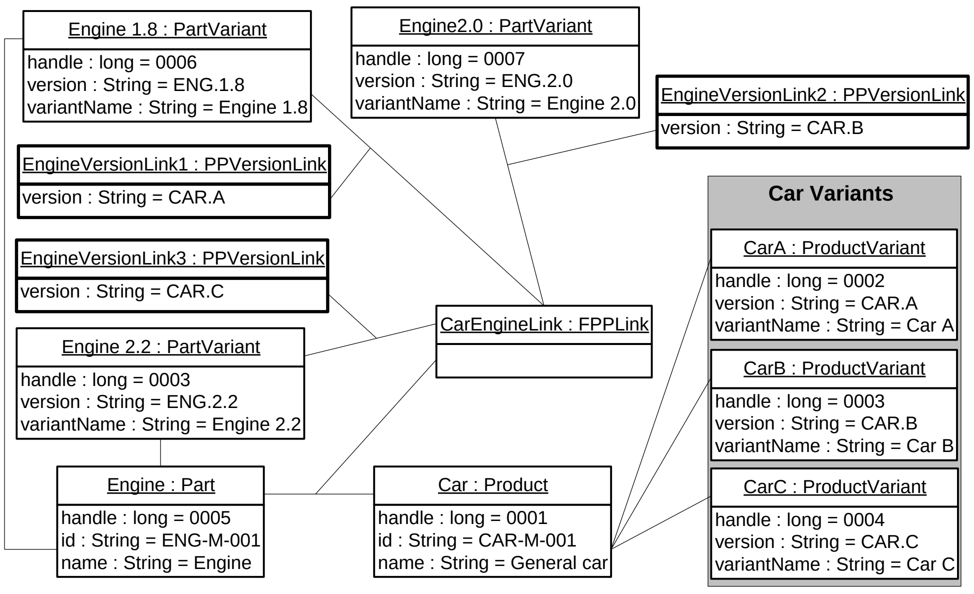


Figure 41 Relationship between a car variant and an engine variant

7.4 Lifecycle Management Support

7.4.1 Product view model

Users with different disciplines usually look into products from different perspectives. For example, purchase staff are only interested in the components which are to be purchased from suppliers or outsourced to partners. A production manager may only be concerned with the components which are to be made or assembled internally. In product lifecycle management, in addition to product structure, a product should be represented in different ways to fulfill different needs. These representations should be consistent with the product structure, which completely reflects product constitution and relationships of constitutional components from the perspectives of functions and structures (Fuxin 2005). To fulfill this need, the developed product structure model is extended to support product views. A product view is a hierarchical representation to associate some of components of a product in different ways, to fulfill needs of a specific stage in a product lifecycle. In product view management, an essential requirement is that a product view should be independent of its product structure. However, it should be easily synchronized with product structure. In other words, the product structure of a product should be kept unchanged while constructing product views. Changes to the product structure should be reflected in the product views. As shown in Figure 42, a reference mechanism is adopted to realize product views. A product view, represented by the class *ProductView*, consists of a set of instances of *PartRef* and/or *SubassemblyRef* organized in a hierarchical structure. Since product views are constructed using part references and subassembly references, they are independent of a product structure. However, the reference mechanism enables product views to be linked back to product structure. Synchronization between product views and the corresponding product structure can be achieved. A reference is a pointer which does not contain the actual data of a part or a subassembly. Therefore, no duplications of data exist and data consistency can be easily maintained. A product can have multiple views, such as manufacturing view, bill of material (BOM) view and engineering change (EC) view. As shown in Figure 42, the categorization of product views is realized based on view roles, which is represented by the link class *ViewRole*. The ability to support product views enables the model to better support product lifecycle management.

7.4.2 Integration with other processes

As shown in Figure 38, the model differentiates standard parts and non-standard parts. The main reasons for differentiating non-standard parts and standard parts are: 1) family concept is inapplicable to standard parts; and 2) processes that non-standard parts go through are different from those of standard parts. Standard parts are purchased from suppliers and managed in inventory. However, non-standard parts may go through various processes, such as a production process if they are made internally or an outsourcing process if they are made by partners, and an inventory management process if they are made to stock.

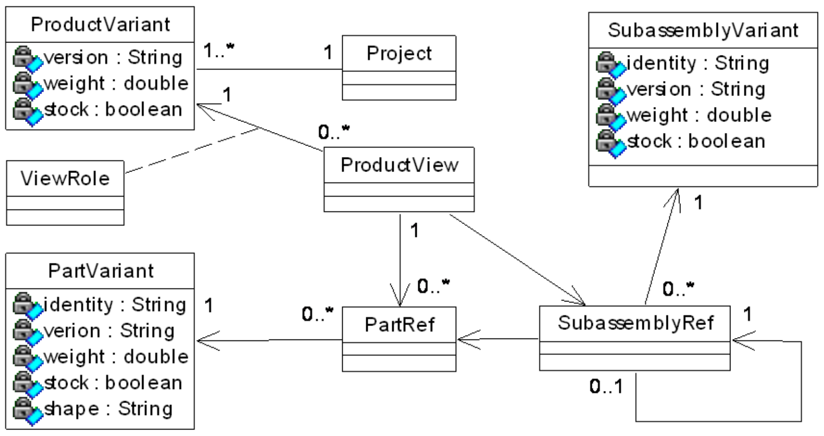


Figure 42 Product view model.

The interfaces *IStockable*, *IPurchasable* and *IOutsourcable* are modeled to enforce the implementing classes to comply with the processing rules of stock management, purchase management and outsourcing management. The implementation of *IStockable* by variant classes, i.e. *ProductVariant*, *PartVariant* and *SubassemblyVariant*, implies: 1) common parts, subassemblies and even products are allowed to be made to stock; and 2) it enables make-to-order and make-to-stock decisions to be made at a variant level. As a result, in a part or subassembly family, variants commonly demanded can be made-to-stock while variants only demanded by a few customers may be particularly made when being ordered.

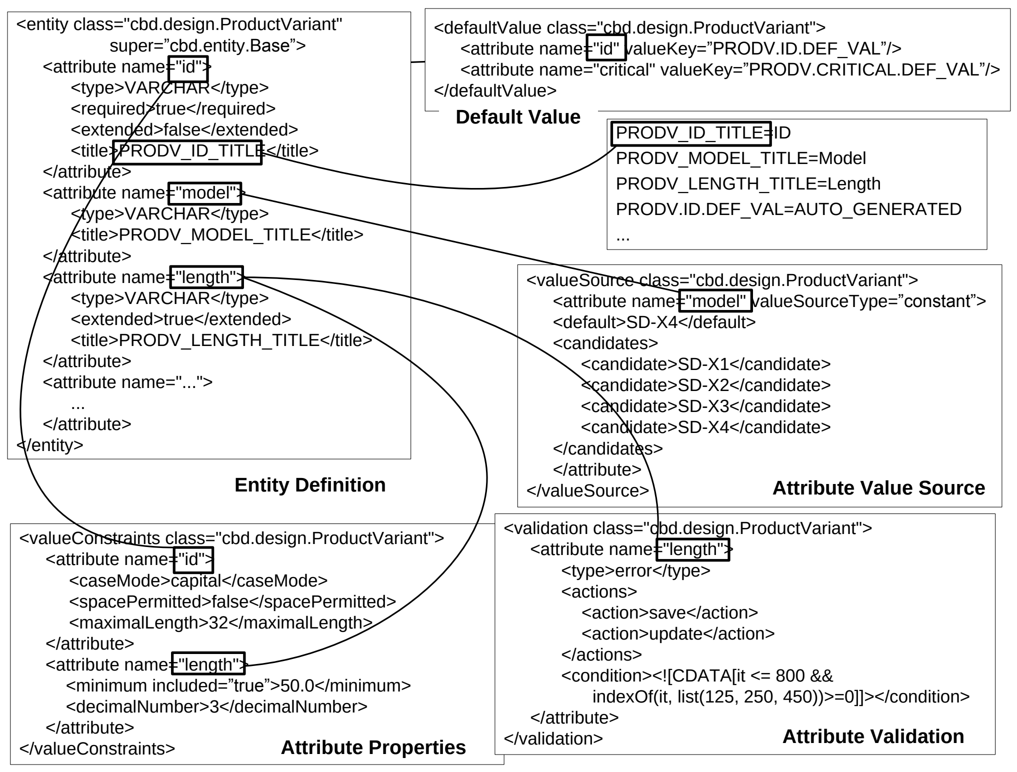
7.5 **Semantic Product Structure Representation**

The product structure model discussed above is abstract and lacks the ability to differentiate and characterize different types of products, parts and subassemblies. To make it useful to the development of PLM systems, the model has to be converted to a concrete model according to industrial sectors. The object-oriented approach to derive a concrete model, based on an abstract model, is called generalization, which is a process to define subclasses by extending abstract classes to represent specific types. For example, the subclasses *Shaft* and *Gear* may be defined by extending the abstract class *Part* to represent shafts and gears, which are more concrete types of parts. However, companies in different industrial sectors have different types of products, parts and subassemblies. Even companies in the same industrial sector may categorize these items in different ways due to the difference of business practices. For example, one company may categorize gears as standard gears and non-standard gears, while another as cylindrical gears and conical gears. The identification of subclasses and the essential attributes of each subclass are difficult at the stage of creating a concrete product structure model. A concrete product structure model developed by creating subclasses may not be sharable to different companies. Traditionally, concrete models are usually established at the design stage and are physically built into a PLM system. In such a way, any changes to a model will cause changes to system source code. Such a PLM system cannot be reused for different companies and lacks the flexibility to support business practice changes.

This case study uses semantic representations to develop a concrete product structure model and make a product structure model loosely coupled with system programs. In such a PLM system, the product structure model can be changed or replaced for different business practices. Such a PLM system is highly flexible and can be easily deployed to different companies, even in different industrial sectors.

7.5.1 Entity Representation

Figure 43 shows the semantic model of product. Entity definition declares all attributes a product entity has. Default value defines default values for the attributes *id* and *critical*. Value sources define value candidates for the attribute *model*. Attribute properties define constraints for the attributes *id* and *length*. Attribute validation contains rules for validating values assigned to the attribute *length*. It indicates that validation is only to be done for two actions: save and update. It can be seen that semantic representations can effectively represent entities in product structures. The representation is similar to and consistent with the machine representation. This proves that the semantic model can be easily constructed.



```
<entity class="cbd.design.ProductVariant"
            super="cbd.entity.Base">
    <attribute name="id">
        <type>VARCHAR</type>
        <required>true</required>
        <extended>false</extended>
        <title>PRODV_ID_TITLE</title>
    </attribute>
    <attribute name="model">
        <type>VARCHAR</type>
        <title>PRODV_MODEL_TITLE</title>
    </attribute>
    <attribute name="length">
        <type>VARCHAR</type>
        <extended>true</extended>
        <title>PRODV_LENGTH_TITLE</title>
    </attribute>
    <attribute name="...">
        ...
    </attribute>
</entity>
```

```
<defaultValue class="cbd.design.ProductVariant">
    <attribute name="id" valueKey="PRODV.ID.DEF_VAL"/>
    <attribute name="critical" valueKey="PRODV.CRITICAL.DEF_VAL"/>
</defaultValue>
```

```
PRODV_ID_TITLE=ID
PRODV_MODEL_TITLE=Model
PRODV_LENGTH_TITLE=Length
PRODV.ID.DEF_VAL=AUTO_GENERATED
...
```

```
<valueSource class="cbd.design.ProductVariant">
    <attribute name="model" valueSourceType="constant">
    <default>SD-X4</default>
    <candidates>
        <candidate>SD-X1</candidate>
        <candidate>SD-X2</candidate>
        <candidate>SD-X3</candidate>
        <candidate>SD-X4</candidate>
    </candidates>
    </attribute>
</valueSource>
```

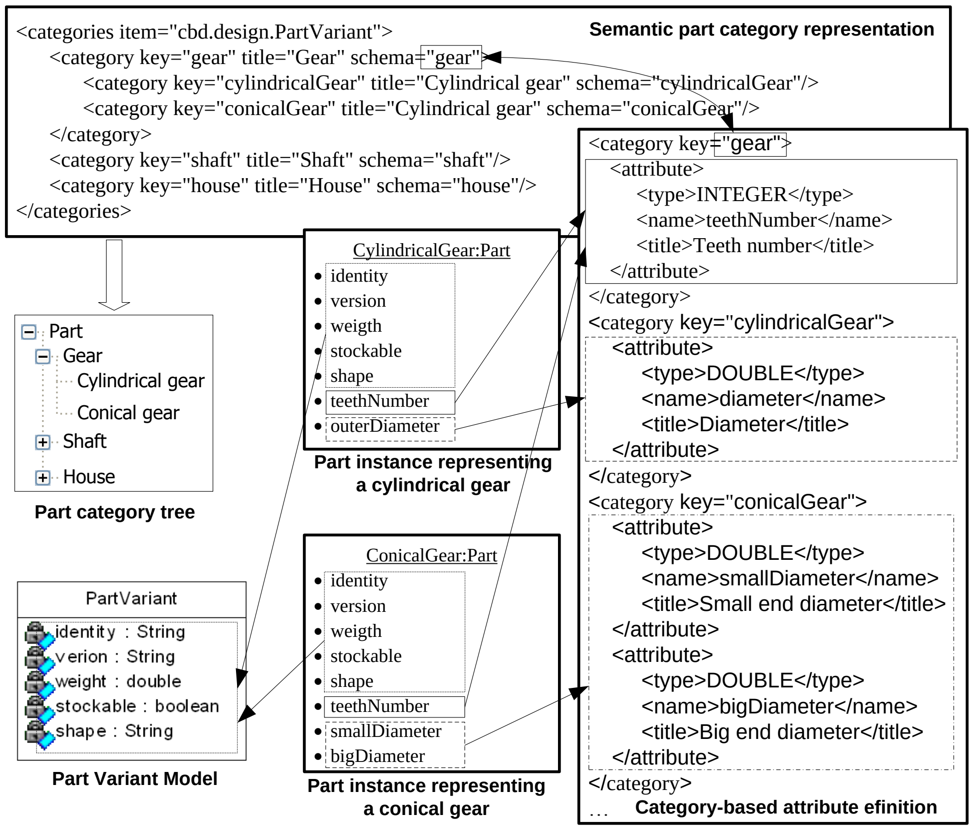
```
<valueConstraints class="cbd.design.ProductVariant">
    <attribute name="id">
        <caseMode>capital</caseMode>
        <spacePermitted>false</spacePermitted>
        <maximalLength>32</maximalLength>
    </attribute>
    <attribute name="length">
        <minimum included="true">50.0</minimum>
        <decimalNumber>3</decimalNumber>
    </attribute>
</valueConstraints>
```

```
<validation class="cbd.design.ProductVariant">
    <attribute name="length">
        <type>error</type>
        <actions>
            <action>save</action>
            <action>update</action>
        </actions>
        <condition><![CDATA[it <= 800 &&
            indexOf(it, list(125, 250, 450))>=0]]></condition>
    </attribute>
</validation>
```

Figure 43 Semantic product definition.

7.5.2 Categorization Representation

The approach of creating different subclasses to represent different types of products, parts and subassemblies results in a rigid concrete model. Therefore, this case study uses semantic category representations for flexible categorization. As shown in Figure 44, semantic category representation organizes categories in a hierarchical format. The tag *category* defines a category using three parameters: *key*, *title* and *schema*. The attribute *schema* contains a keyword pointing to a group of attribute definitions associated with the category. A collective category, such as gear, can have sub-categories, such as cylindrical gear and conical gear, which are represented as nested elements of the collective category. The attributes defined for a collective group will be inherited by all its sub-categories. Apart from attributes defined in the class *PartVariant*, instances of *CylindricalGear* and *ConicalGear* also have attribute *teethNumber* which is defined for the category *gear*. At the same time, *CylindricalGear* and *ConicalGear* instances have specific attributes respectively to characterize cylindrical gears and conical gears. This approach does not require identifying all subclasses at a design stage because categories and category-related attributes can be configured in semantic models.



```
<categories item="cbd.design.PartVariant">
    <category key="gear" title="Gear" schema="gear">
        <category key="cylindricalGear" title="Cylindrical gear" schema="cylindricalGear"/>
        <category key="conicalGear" title="Cylindrical gear" schema="conicalGear"/>
    </category>
    <category key="shaft" title="Shaft" schema="shaft"/>
    <category key="house" title="House" schema="house"/>
</categories>
```

```
<category key="gear">
  <attribute>
     <type>INTEGER</type>
     <name>teethNumber</name>
     <title>Teeth number</title>
  </attribute>
</category>
<category key="cylindricalGear">
  <attribute>
     <type>DOUBLE</type>
     <name>diameter</name>
     <title>Diameter</title>
  </attribute>
</category>
<category key="conicalGear">
  <attribute>
     <type>DOUBLE</type>
     <name>smallDiameter</name>
     <title>Small end diameter</title>
  </attribute>
  <attribute>
     <type>DOUBLE</type>
     <name>bigDiameter</name>
     <title>Big end diameter</title>
  </attribute>
</category>
...
```

Figure 44 Semantic category representation.

7.6 Summary

In the manufacturing industry, product structure needs to be managed to support various business processes. Product structure representation is critical to PLM systems. In a make-to-order environment, a product is a family with a number of variants rather than a single product. Accordingly, a product structure representation should be able to characterize common characteristics and particular characteristics of individual variants from a different perspective. It is a significant industrial need to develop a product structure model that can effectively represent the family structure and variant structures, and support different stages of the product lifecycle. This case study adopts a master-variant pattern for establishing such a product structure model. To make the product structure model extensible, semantic representations are applied to extend the abstract product structure model to a concrete model for developing flexible PLM systems. This case study proves that semantic representation can be used to represent comprehensive business information models with complicated hierarchical structures. Compared to the previous chapter, the semantic machine model and semantic product model are very similar. This will also prove that semantic representations are consistent and easy to use.

Chapter 8 Semantic Reporting Modeling

8.1 Introduction

This chapter develops a flexible reporting model to verify the usability of semantic representations in cases where massive and complicated information processing is involved. According to the architecture of model driven enterprise systems, the reporting function is separated into two parts: report configurations and software programs. Report configurations are based on semantic representations and contain instructions that guide computer programs to generate reports. By providing different report configurations, the same software program can generate different types of reports. This approach enables the development of a generic and flexible reporting solution which can be reused in different enterprises.

8.2 Background

The purpose of an enterprise system is to provide the right person with the right information in the right format at the right time by capturing, generating, associating, populating information according to business practices (Clive and Aiken 1999). To effectively assist end users, information must be presented in a structured and concise format. Indeed, reports play a very important role in enterprise systems. Reports provide summarized information about the status of resources, progress of jobs and profit profiles for planning and decision-making. Reports also contribute to communication with external partners, e.g. suppliers and customers. There are many types of reports to be generated and managed to facilitate operational management in enterprises. Based on the knowledge obtained from our industrial partners, there typically exist 50 to 80 types of reports in small and medium-sized enterprises (SMEs). Therefore, reporting is one of the critical functions in enterprise systems and it has been one of the key functionalities to be evaluated in the selection of enterprise systems.

Different types of reports usually have different appearances and contents. Even for the same type of reports, the appearances and contents can also vary from one company to another. After a system is deployed, end users may have the need to adjust report formats and/or contents, and to add new reports due to changes in business practices. Though the need of a flexible reporting method has existed for a long time, it has received little research attention (Kahn 1998, Jensen and Baumgartner 2003). This research presents a configuration-based reporting method that can be used to develop flexible reporting solutions. Such solutions can be rapidly customized to satisfy the requirements of different companies. They can also be easily reconfigured to support new business practices even after being deployed. As a result, the deployment cycle of an enterprise system can be effectively shortened and end users can have more freedom to change their business practices when necessary.

A few studies on reporting can be found in the literature (Kahn 1998, Langlotz 2000b, Jensen, et al. 2003, Luo and Bai 2005). These studies only discussed report generation for some specific applications without consideration of flexible reporting methods for enterprise information systems. Langlotz (Langlotz 2000a) carried out a study on structured reporting for radiology practices. This study came out with a structured model that enabled the embedding of multimedia into reports and made report contents semantic. Reports based on the model could be easily shared through the internet and stored to support future research. Jensen (Jensen, et al. 2003) also reported a structured reporting system for hospital use. The system adopted a template-based method for report generation. The templates were represented as XML documents. Extensible stylesheet language (XSL) was adopted to transform the template for displaying on screen for data collection. The collected data was converted to structured report objects based on the standard of digital imaging and communications (DICOM) in medicine. Based on the Web services technology, Luo (Luo, et al. 2005) developed a reporting system for generating software test reports. The system is composed of three components:

report definition, report generator and report presentation. The report definition was employed to define the types of reports and database tables and table fields to be shown in the report. Similar to the system reported by Jensen, XSL was also used to control report visualization.

The above efforts focused on modeling and designing reporting systems for specific applications. Reporting functions for enterprise systems are usually much more complicated. The information shown in enterprise reports is not directly collected from end users with an empty report on screen. Instead, most information is processed by other functional modules and stored in different database tables. Furthermore, many enterprise systems make the structures of, and relationships between, database tables transparent to implementers and end users (Keller and Teufel 1998, PTC 2000). Implementers and end users may have difficulties in understanding the database tables and relationships between these tables. Reports cannot be generated directly based on database tables.

Reporting solutions provided by commercial enterprise systems are also not powerful and flexible enough (Henschen 2005). Many companies with enterprise systems have complained, citing difficulties for their management in finding out how the business was performed, because not enough formal printed reports were provided by the adopted enterprise systems (Ross, et al. 2000). These systems put focus on managing online transactions rather than converting data into information to assist in making decisions. In addition, the report solutions provided by these enterprise systems usually store structured report data in a database with limited configurability. Contents in reports cannot be changed without the modification of system source code (Jensen, et al. 2003). Hampered by limited technologies, redesign and redevelopment are dominant approaches in delivering tailored reporting solutions for companies (Kahn 1998). Therefore, this case study develops a reporting method based on semantic representations.

8.3 **Overview of the Reporting Method**

The reporting method proposed facilitates the development of generic reporting functions which can be easily adapted to different companies through semantic model configuration with little redesign and redevelopment. As shown in Figure 45, the proposed method decouples report generation logic from the report generation program. The report generation program is not developed based on generation logic of individual reports. Instead, it is designed to create reports by interpreting generation logic in semantic models. Accordingly, based on the architecture of a model-driven enterprise system, such a reporting solution consists of two components: semantic reporting models and a generation program. Generation logic is represented in semantic reporting models so that generation logic can be changed easily. As the report generation program is driven by generation logic explicitly represented in semantic reporting models, the program can produce different reports by providing different sets of models. When reporting models related to a report are modified, the newly generated reports will be different from the ones that were generated previously. Instead of writing different software programs to generate different reports, different sets of semantic reporting models can be composed for different reports. Similarly, to modify an existing report, the method is to revise corresponding models rather than to modify the generation program. The customization of such a report solution is to construct different sets of semantic reporting models while the generation program can be kept unchanged.

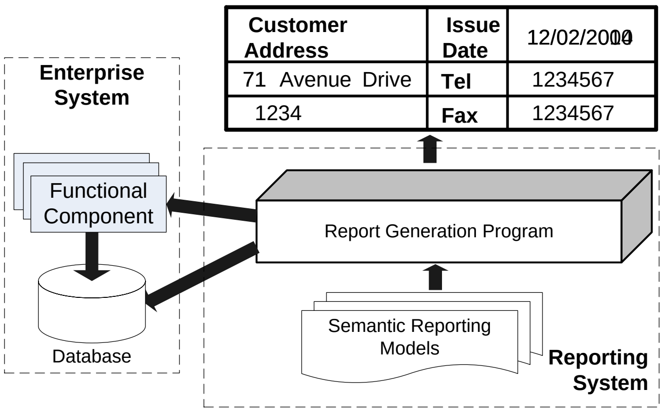


Figure 45 Overview of the method

8.4 Semantic Reporting Model

The flexibility of the report generation is achieved by utilizing different types of semantic models which decouple report generation logic from the report generation program. This section details semantic reporting models which integrate various logic representations to achieve flexible report generation. As the structure of reports plays an important role in semantic reporting model construction, structural analysis is taken as an entry point to discuss semantic reporting models. To be generic, a report may consist of multiple pages, including a cover page and/or an end page. Different pages may have different formats. Each page of a report, including the cover page and/or end page if they are presented, can be logically divided into different data areas, optionally separated by lines in different styles. Furthermore, a data area may constitute various cells with different sizes and styles. The cells can be located using row and column indices. According to this structural breakdown, each page of a report can be deemed as various information entries shown at different locations in different styles.

Based on the structural decomposition, the semantic reporting representation framework shown in Figure 46 is developed for configuring various types of reporting logic. In general, the differences of reports can exist in formats, such as appearances, information entries, entry locations and entry display styles. To achieve a flexible reporting solution, report formats should be configurable and information entries in reports should be able to be redefined easily. As shown in Figure 46, the framework employs a report type model, template model, object acquisition model, and entry imposition model to realize flexible configuration. Of these semantic models, the report type model is used to define report types. The report template defines report appearances and data areas. The object acquisition is introduced to configure the logic used to retrieve related objects, which are to be processed to derive information entries. The imposition model is employed to define the logic of determining information entries and the locations of each entry. The positions and display styles of information entries can also be specified in this model. Since report types, report formats, information entries, and entry positions and entry display styles can be flexible configured, a reporting solution is flexible and configurable.

In companies, various types of reports need to be generated and managed to facilitate business operations and decision making. Furthermore, report types can vary from one company to another. The report type model provides a means for defining report types needed by individual companies. By revising this model, new report types can be added and unwanted report types can be removed. Therefore, through the report type model, a tailored reporting environment can be easily achieved for a particular company.

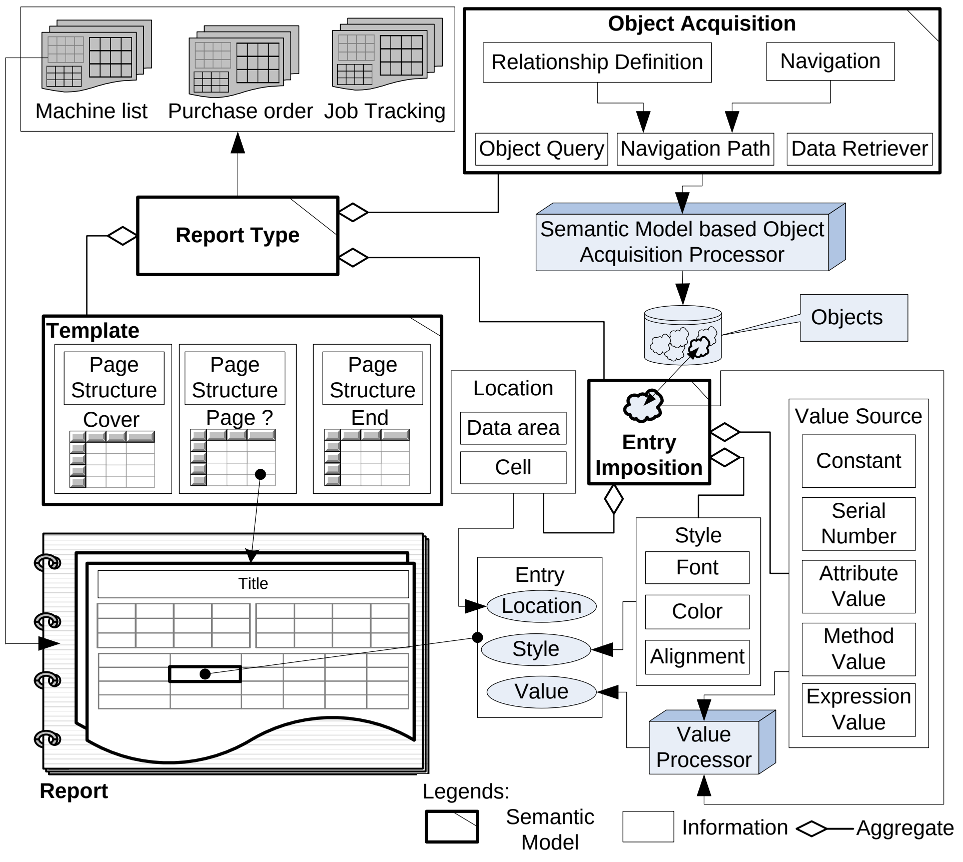


Figure 46 Semantic Reporting Representation Framework

A type of report can be uniquely characterized by the format, information entries, entry locations and entry display styles. The proposed method clusters report generation logic into three types of models: templates, object acquisition and entry imposition. The report templates are utilized to define report appearances, such as borders and grid lines, and describe the structures, such as data areas and the arrangement of data areas. As shown in Figure 46, appearances and structures are defined on a page basis. As such, different pages can have different appearances and structures. This should also allow multiple pages to share a template to minimize the effort of semantic model construction. The positions of information entries in a data area can be specified in two ways: absolute and relative. The absolute way is to specify the absolute indices of row and column according to the upper-left corner of a data area and can only be used for independent information entries. On the other hand, if the position of an information entry can only be worked out when the positions of a related entry are determined, a relative way has to be used for defining positions of information entries. For example, in a report listing machines based on work centers, the position of machine entries can only be decided after the work center entry is located. Accordingly, the data areas are categorized into absolute data areas and relative data areas. When positioning entries in a relative way, the growing direction, either horizontal or vertical, can be defined. Relative data areas can be classified into vertical areas and horizontal areas. They are also used to control pagination. A page can have multiple relative data areas and a virtual pointer can be assigned to each relative data area to record the place where the next entry should appear. When a pointer is out of the data area, an event is triggered to inform the generation program that a new page should be initialized.

Report generation is a process of deriving information entries by manipulating relevant objects and imprinting information entries to specified data areas of a blank report. It is obvious that the flexibility of a reporting solution is limited if the logic of gathering necessary objects for report generation is physically built into the report generation program. In this case, implementers and end users lose opportunities to change information entries for reports. It is impossible to add new reports without updating system source code. To improve the flexibility of a reporting solution, the logic of object acquisition is represented in semantic models to guide object query, relationship navigation, or invocation of services provided by enterprise systems to gather objects. By separating object acquisition logics from the report generation program, it is made possible to customize the contents of reports without changing the generation program.

Objects needed for report generation are classified into two categories: primary objects and secondary objects. The objects which have to be provided as initial input are referred to as primary objects. The objects which can be dynamically retrieved through relationship navigation based on primary objects are defined as secondary objects. For example, in purchase order report generation, a purchase order object is a primary object since it can only be retrieved by a direct database query. The objects representing order items in a purchase order are secondary objects because they can be navigated based on the semantic relationship model defined between purchase order and order items. Accordingly, the representation framework introduces object query model for retrieving primary objects and navigation model for retrieving secondary objects. The query model provides instructions, which are ultimately converted to SQL statements, to retrieve data from a database to construct primary objects. The navigation configuration provides information about relationships between objects to guide relationship navigation. The model also allows specification of an object retriever for the report generator to gather objects. An object retriever is a plug-and-play component that can be invoked to collect objects. It can be developed by reusing some application functions.

The entry imposition model represents object manipulation logic for deriving information entries, provides instructions for allocating information entries to correct positions and specifies display styles of information entries. The entry imposition model offers opportunities to change information entries, relocate information entries, and redefine display styles. As indicated in the representation framework, information entries shown in reports can be configured as constants, serial numbers, object attribute values, object method values or expression values. Constants, which are usually the titles of information entries, are directly used as information entries. Serial numbers are a set of continuous integers, mostly indices of a collection of related information entries. Attribute values indicate using attribute values as information entries. Similar to the attribute values, method values instruct a generation program to invoke the methods on objects to obtain information entries. Expression values represent complex logic in manipulating relevant objects to derive information entries.

To effectively facilitate report generation, objects provided for report generation should be associated in an appropriate way so that the objects can be easily and quickly found when needed. By addressing these issues, the object association model has been developed. Figure 47 shows the structure by using a purchase order as an example. The model organizes objects in a hierarchical tree and identifies object using names, which are defined in semantic models. In the model, each node at the first level holds a keyword, which is the name of object(s) held by its child nodes. Recursively, an object node can further have name nodes to hold a set of objects. For example, the object node person has a name node address to hold an object node of *Address*. It means that the object *Address* is associated with the object *Person* to represent the person's address.

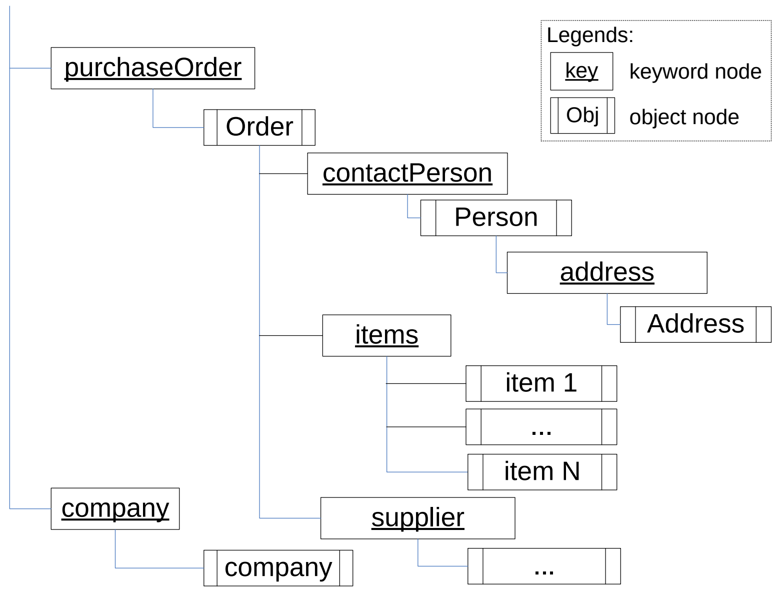


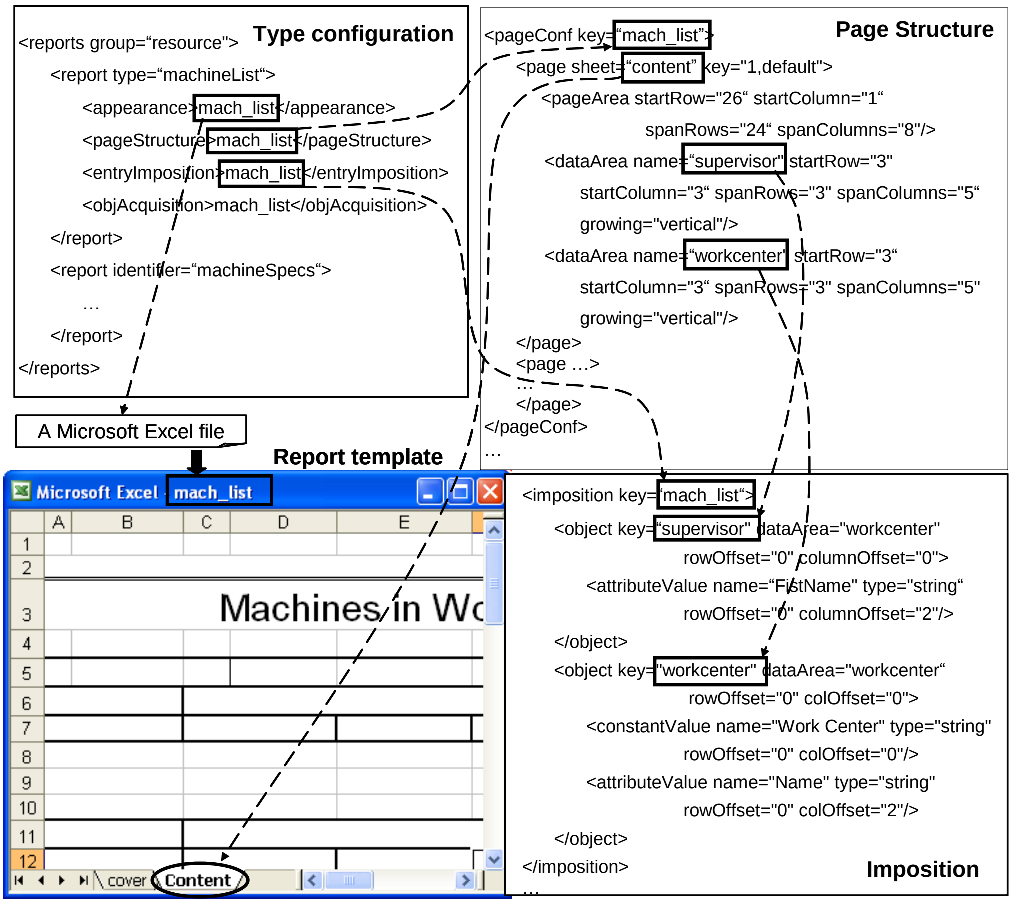
Figure 47 Object association model

8.5 Semantic Reporting Configuration Language

A semantic and easy-to-use configuration language is essential to enable implementers and end users to undertake reporting configuration. The syntaxes of the semantic reporting configurations are illustrated in Figure 48. Report types, page structures, imposition and object acquisition are configured in separate files. In the type configuration, appearance, *pageStructure*, *entryImposition* and *objAcquisition* are four tags for linking a report type to its appearance template, page structure model, imposition model and object acquisition model.

8.5.1 Report type Model

Each report type is defined by the tag *report* with four nested tags, which are *appearance*, *pageStructure*, *objAcquisition* and *entryImposition*. The tags of *appearance*, *pageStructure* and *entryImposition* are compulsory and the tag *objAcquisition* is optional. Each nested tag defines a value as a keyword pointing to corresponding models. The report type model can be used to dynamically lay out a reporting environment for end users. A specific reporting environment can be easily provided for individual companies.



```
<reports group="resource">
    <report type="machineList">
        <appearance>mach_list</appearance>
        <pageStructure>mach_list</pageStructure>
        <entryImposition>mach_list</entryImposition>
        <objAcquisition>mach_list</objAcquisition>
    </report>
    <report identifier="machineSpecs">
        …
    </report>
</reports>
```

```
<pageConf key="mach_list">
  <page sheet="content" key="1,default">
    <pageArea startRow="26" startColumn="1"
              spanRows="24" spanColumns="8"/>
    <dataArea name="supervisor" startRow="3"
        startColumn="3" spanRows="3" spanColumns="5"
        growing="vertical"/>
    <dataArea name="workcenter" startRow="3"
        startColumn="3" spanRows="3" spanColumns="5"
        growing="vertical"/>
  </page>
  <page …>
  …
  </page>
</pageConf>
…
```

```
<imposition key="mach_list">
   <object key="supervisor" dataArea="workcenter"
          rowOffset="0" columnOffset="0">
      <attributeValue name="FirstName" type="string"
          rowOffset="0" columnOffset="2"/>
   </object>
   <object key="workcenter" dataArea="workcenter"
          rowOffset="0" colOffset="0">
      <constantValue name="Work Center" type="string"
          rowOffset="0" colOffset="0"/>
      <attributeValue name="Name" type="string"
          rowOffset="0" colOffset="2"/>
   </object>
</imposition>
…
```

Figure 48 Semantic report configuration

8.5.2 Template Model

The template model provides the information of report appearances and structures on a page basis. Providing a visual environment for defining or changing report appearances can make the appearance definition much easier and more efficient. In this case study, Microsoft Excel is selected for this purpose as it is very popularly used. It needs to be pointed out that Excel spreadsheets only play a role of defining the formats of reports. Contents, entry imposition and information processing logic are specified in other models. In such a way, the report formats can be changed independently and safely without any potential damages to other models.

On the top of the appearance model, the page structure model is introduced to further describe page structures from two aspects: page area and data area. The page area is the effective area of a Microsoft Excel spreadsheet where a page appearance is defined. The tag *pageArea* is used to define page areas, which has the compulsory attributes *startRow*, *startColumn*, *spanRows* and *spanColumns*. The attributes *startRow* and *startColumn* define the upper-left corner of the effective area and *spanRows* and *spanColumns* specify the size of the effective area in row and column. The effective page area is divided into different data areas. The tag *dataArea* is employed for defining data areas of a page. Each data area is assigned a unique name for reference. Similar to the tag *page*, the tag *dataArea* has four attributes, i.e. *startRow*, *startColumn*, *spanRows* and *spanColumns*, for defining the ranges of data areas. The difference is that the upper-left corner of a data area is relative to the upper-left corner of the page area while the page upper-left corner is defined using the absolute indices of row and column in a spreadsheet.

The tag *page* has two required attributes: *sheet* and *key*. The attribute *sheet* associates the structure model with a spreadsheet in a workbook of Microsoft Excel, which defines appearance of the page. The attribute *key* indicates which pages the model is defined for. The value of the attribute *key* is a page number or combination of multiple page numbers. For multiple page numbers, attribute values can be given in three formats: 1) enumeration, such as “1,2,3”; 2) range, such as “1-3”; or 3) hybrid, a combination of enumeration and range, such as “1,2,4-6”. The keywords of “cover”, “end”, “even” and “odd” are reserved for the cover page, end page, even pages and odd pages respectively. The keyword “default” is also reserved for pages without a page structure model specified. In other words, when no page structure model is found based on a page number, the page model whose attribute *key* is set to “default” is automatically selected. The rule for finding a page structure model is described as follows. Firstly, the page number is used to search for a page structure model. If not found, then, a proper keyword of “cover”, “end”, “even” or “odd” is used for further searching. Finally, the keyword “default” is used to search if no page structure is found in the previous two steps.

8.5.3 Imposition Model

The imposition model represents the logic of deriving information entries and positioning entries. Entry display styles can also be specified in this model. In the tag *object*, an object and the logic can be specified. The object is to be processed according to specified logic to derive an information entry. The attribute *dataArea* refers to the data area where the derived entry should appear. The tag *object* has five nested tags, which are *constantValue*, *serialNumber*, *attributeValue*, *methodValue* and *expressionValue*, to define logic for deriving information entries. The display styles, such as font, alignment and line break, can be configured in the tag object and the nested tags. The styles specified in the tag *object* take effect for all the nested tags. The styles defined in a nested tag overwrite the styles defined in the tag *object* and only take effect for that nested tag.

For interdependent information entries, such as the work centre-based machine list report stated in the previous section, a hierarchical format is used to configure the interdependent entries. As illustrated in Figure 49, the second *object* tag is nested in the first one. It implies that the positions of machine entries are relative to the positions of the corresponding work centers.

```
<imposition>
   <object key="workcenter" dataArea="workcenter" rowOffset="0" …>
      <constantValue name=“Work Center" type="string" rowOffset="0" …/>
      <attributeValue name=“name" type="string" rowOffset="0" ../>
      …
      <object key="machine" dataArea="workcenter" rowOffset="0" …>
          <attributeValue name=“id" rowOffset="1" columnOffset="0"/>
          <attributeValue name=“title" rowOffset="0" columnOffset="2"/>
          …
      </object>
   </object>
</imposition>
```

Figure 49 Imposition configuration of a structured report

8.5.4 Object Acquisition Model

The object acquisition model is incorporated to prevent object acquisition logic from being hard-coded into the report generation program. It maximizes the ability to accommodate potential changes to report contents. As stated above, three types of object acquisition logic can be configured, which are object query, relationship navigation and object retriever. The object retriever is a plug-and-play component for retrieving objects. Object query, relationship navigation and expression have been discussed in detail in Chapter 4.

8.6 Summary

This case study has outlined a reporting method for developing flexible reporting solutions based on semantic representations. In this method, templates, page structure model and imposition model are employed for defining types of reports and the logic of report generation. The report templates define appearances and structures of individual pages of reports. The page structure model further describes the structural constitution of reports based on the appearance definition. The imposition model contains the logic of deriving, positioning and displaying information entries. To maximize this flexibility, three types of object acquisition models are employed. The object acquisition models play the role of preventing object acquisition logic from being physically coded into the report generation program. An object association model has also been established to organize objects in a tree format to effectively support report generation.

The method provides the following flexibility with the support of semantic representations:

- Providing a semantic and systematic approach for defining report generation logic;
- Semantic report generation logic representation makes the construction and modification of report configurations independent of any specific tools and platforms;
- Enabling implementers and end users to easily redefine report formats and change information entries to be shown on reports;
- Providing opportunities for implementers and end users to configure new reports and to remove unwanted reports;

- Reusing or sharing information processing capabilities of enterprise information systems.

This case study demonstrates the ability of semantic models to represent complex business logic. It proves that semantic representations can be used in cases when complicated processing logic is involved.

Chapter 9 PROTOTYPE

9.1 Introduction

This chapter develops a proof-of-concept system to study the process of the development of model driven enterprise systems and further verify the effectiveness of model driven enterprise systems. The prototype is developed in Java. The apache web server is adopted as the web server and Tomcat is selected as the JSP (Java ServerPage) engine. The functions of programmatically processing Microsoft Excel files are developed using APIs developed by the Apache POI (Poor Obfuscation Implementation) project, which aims to develop a whole set of Java-based APIs for manipulating various file formats based upon Microsoft's OLE 2 Compound Document (Oliver, Stampoultzis and Sengupta 2004).

9.2 Object Model Integration

System models developed in the three case studies are discussed separately. These models are integrated and linked to the configurable object model in order to incorporate semantic models, as illustrated in Figure 50. The configurable object model provides the ability to map semantic to memory objects. It also establishes a connection between semantic models and business object models. In the configurable object model, the class *EntityDefinition* has an aggregation association with the class *AttributeDefinition*. The class *EntityDefinition* represents semantic entity models and the class *AttributeDefinition* represents semantic attribute declarations. This implies that a semantic entity model is a collection of semantic attribute definitions. The classes *AttributeValueSource*, *ValueConstraints* and *AttributeValidation* are associated with the class *AttributeDefinition* so that information about attribute value candidates, attribute value constraints and validation rules are liked to entity objects through the class *EntityDefinition*.

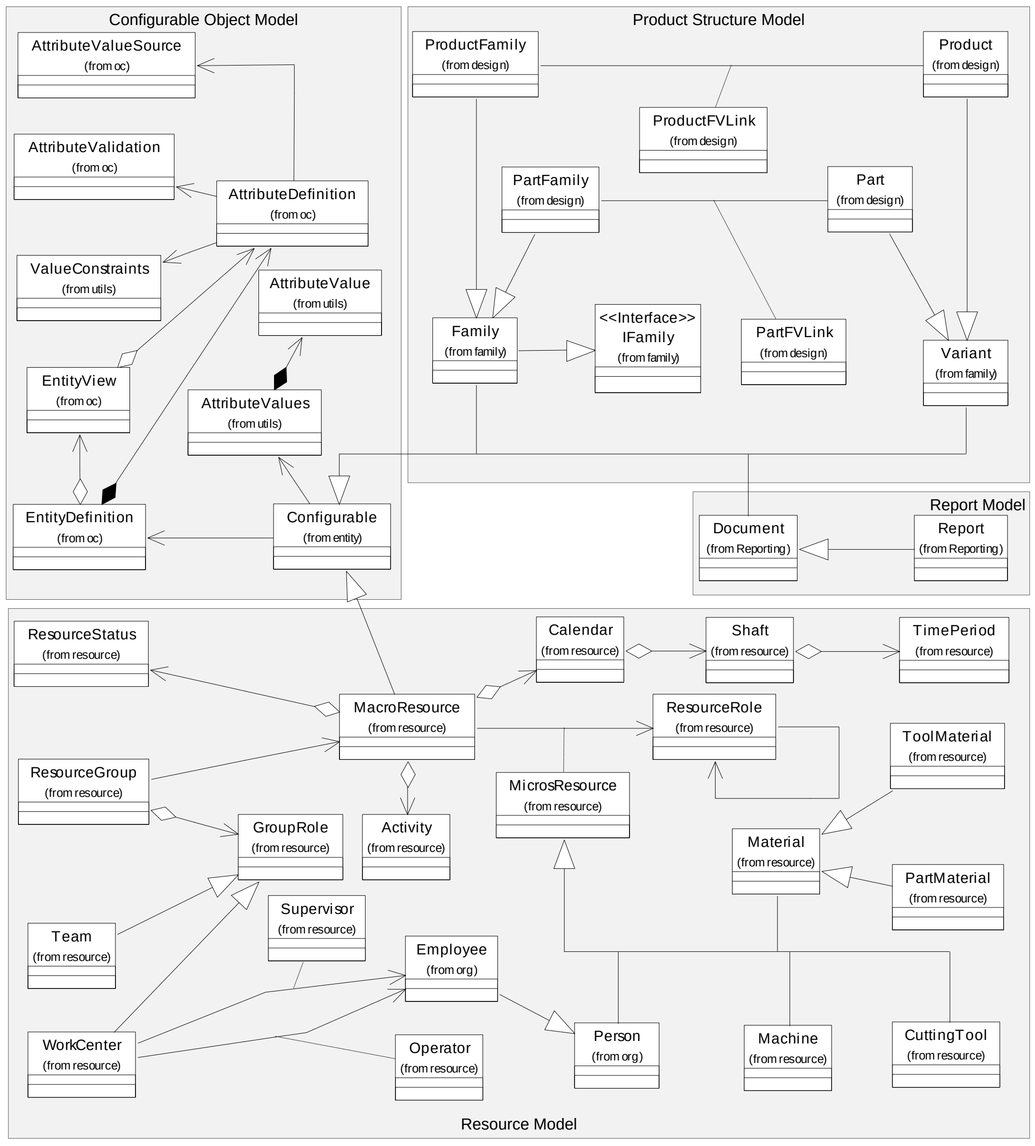
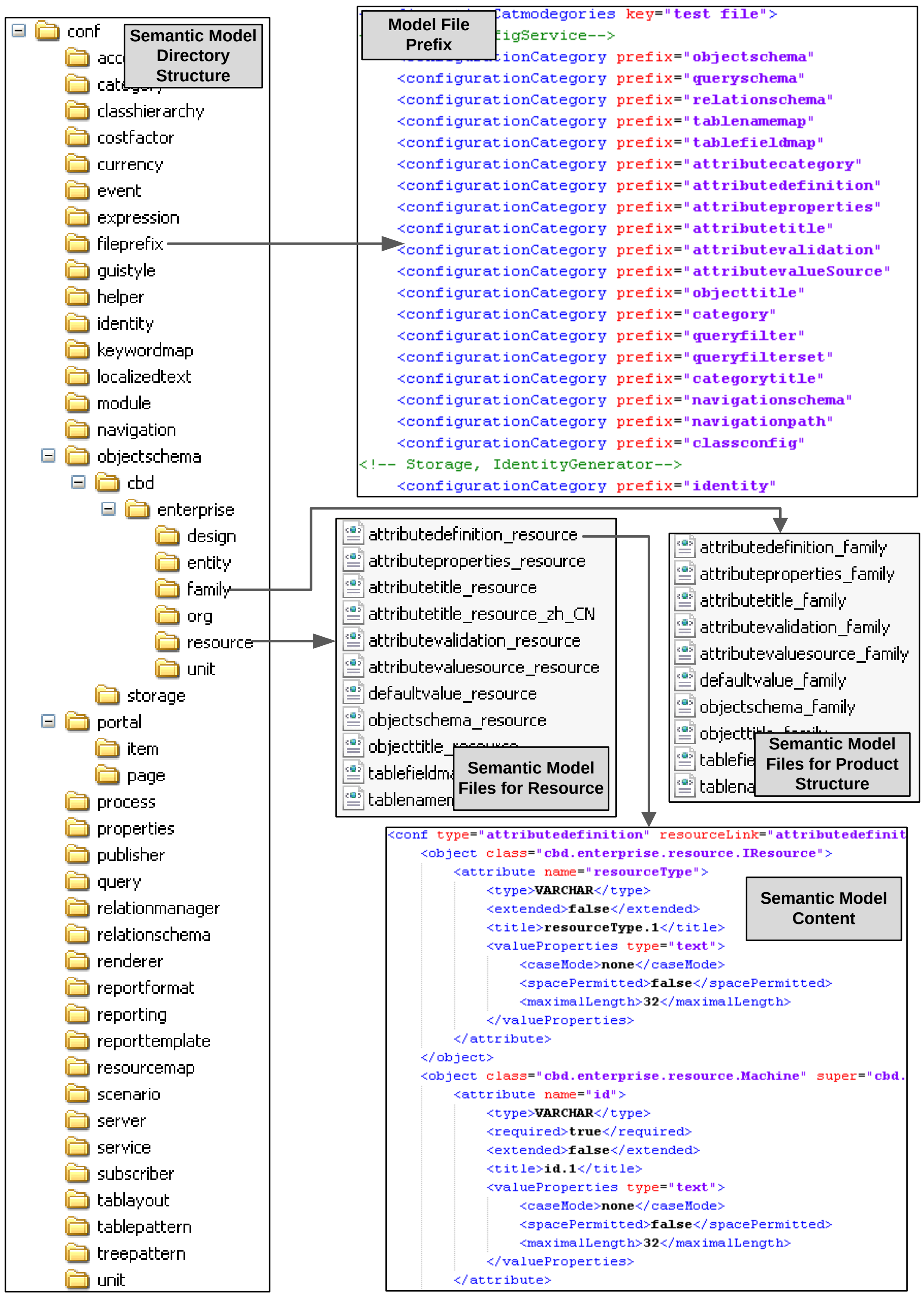

Configurable Object Model
AttributeValueSource (from oc)
AttributeValidation (from oc)
AttributeDefinition (from oc)
ValueConstraints (from utils)
AttributeValue (from utils)
EntityView (from oc)
AttributeValues (from utils)
EntityDefinition (from oc)
Configurable (from entity)
Product Structure Model
ProductFamily (from design)
Product (from design)
ProductFVLink (from design)
PartFamily (from design)
Part (from design)
Family (from family)
<<Interface>> IFamily (from family)
PartFVLink (from design)
Variant (from family)
Report Model
Document (from Reporting)
Report (from Reporting)
ResourceStatus (from resource)
Calendar (from resource)
Shaft (from resource)
TimePeriod (from resource)
MacroResource (from resource)
ResourceRole (from resource)
ToolMaterial (from resource)
ResourceGroup (from resource)
MicrosResource (from resource)
GroupRole (from resource)
Activity (from resource)
Material (from resource)
PartMaterial (from resource)
Supervisor (from resource)
Team (from resource)
Employee (from org)
WorkCenter (from resource)
Operator (from resource)
Person (from org)
Machine (from resource)
CuttingTool (from resource)
Resource Model

Figure 50 Model integration

The class *Configurable* has an association with the class *EntityDefinition*. Therefore, each configurable entity, which is defined as a direct or indirect subclass of the class *Configurable*, has a semantic model associated. The class *Configurable* also has an association with the class *AttributeValues*, which can hold a set of attribute values because it has an aggregation relationship with the class *AttributeValue*. The class *EntityView* enables the definition of a subset of attribute definitions as a view. For the integration view, it can be observed that the class *Configurable* is a bridge between semantic models and business object models. All business entity classes in the resource model, product structure model and report model are directly or indirectly extended from the class *Configurable*. Consequently, these entities inherit the ability from the class *Configurable* to support model driven configuration.

9.3 **Semantic Model Organization**

To achieve model driven enterprise systems, many system models need to be represented as semantic models. In nature, these models are various XML documents. These models have to be well organized and associated together so that they can be effectively managed and used to drive the behavior of software programs. This research employs the directory structure to organize semantic model files, as shown in Figure 51. The *conf* directory contains a set of subdirectories. A subdirectory may have its own subdirectories. Semantic model files are organized into different subdirectories according to their logical relevance. For example, this prototype organizes all semantic models for business objects into the subdirectories of the *enterprise* directory. However, different enterprises may have different preferences to organize these model files. In order to provide flexibility for enterprises to freely design this structure, a model file name configuration is introduced to achieve this goal. The model file name configuration is an XML document which defines the name prefixes of semantic model files. For example, the name of entity model files starts with *entitydefinition*; and the name of semantic attribute definition files starts with *attributedefinition*. As shown in Figure 51, both attributedefinition_resource.xml and attributedefinition_family.xml (the suffix .xml is not displayed) are semantic attribute definition files. When parsing semantic models, the model parser recursively scans all subdirectories of the *conf* directory. For instance, when parsing attribute definitions, the parser will collect all files with a name starting with *attributdefinition* in any subdirectories of the *conf* directory.


Semantic Model Directory Structure
Model File Prefix
Semantic Model Files for Resource
Semantic Model Files for Product Structure
Semantic Model Content

```
⊟ conf
    acc
    cat
    classhierarchy
    costfactor
    currency
    event
    expression
    fileprefix
    guistyle
    helper
    identity
    keywordmap
    localizedtext
    module
    navigation
  ⊟ objectschema
    ⊟ cbd
      ⊟ enterprise
          design
          entity
          family
          org
          resource
          unit
      storage
  ⊟ portal
      item
      page
    process
    properties
    publisher
    query
    relationmanager
    relationschema
    renderer
    reportformat
    reporting
    reporttemplate
    resourcemap
    scenario
    server
    service
    subscriber
    tablayout
    tablepattern
    treepattern
    unit
```

```
Catmodegories key="test file">
igService-->
ionCategory prefix="objectschema"
    <configurationCategory prefix="queryschema"
    <configurationCategory prefix="relationschema"
    <configurationCategory prefix="tablenamemap"
    <configurationCategory prefix="tablefieldmap"
    <configurationCategory prefix="attributecategory"
    <configurationCategory prefix="attributedefinition"
    <configurationCategory prefix="attributeproperties"
    <configurationCategory prefix="attributetitle"
    <configurationCategory prefix="attributevalidation"
    <configurationCategory prefix="attributevalueSource"
    <configurationCategory prefix="objecttitle"
    <configurationCategory prefix="category"
    <configurationCategory prefix="queryfilter"
    <configurationCategory prefix="queryfilterset"
    <configurationCategory prefix="categorytitle"
    <configurationCategory prefix="navigationschema"
    <configurationCategory prefix="navigationpath"
    <configurationCategory prefix="classconfig"
<!-- Storage, IdentityGenerator-->
    <configurationCategory prefix="identity"
```

```
attributedefinition_resource
attributeproperties_resource
attributetitle_resource
attributetitle_resource_zh_CN
attributevalidation_resource
attributevaluesource_resource
defaultvalue_resource
objectschema_resource
objecttitle_
tablefieldm
tablenamen
```

```
attributedefinition_family
attributeproperties_family
attributetitle_family
attributevalidation_family
attributevaluesource_family
defaultvalue_family
objectschema_family
objectti
tablefie
tablena
```

```
<conf type="attributedefinition" resourceLink="attributedefinit
    <object class="cbd.enterprise.resource.IResource">
        <attribute name="resourceType">
            <type>VARCHAR</type>
            <extended>false</extended>
            <title>resourceType.1</title>
            <valueProperties type="text">
                <caseMode>none</caseMode>
                <spacePermitted>false</spacePermitted>
                <maximalLength>32</maximalLength>
            </valueProperties>
        </attribute>
    </object>
    <object class="cbd.enterprise.resource.Machine" super="cbd.
        <attribute name="id">
            <type>VARCHAR</type>
            <required>true</required>
            <extended>false</extended>
            <title>id.1</title>
            <valueProperties type="text">
                <caseMode>none</caseMode>
                <spacePermitted>false</spacePermitted>
                <maximalLength>32</maximalLength>
            </valueProperties>
        </attribute>
```

Figure 51 Imposition configuration of a structured report

In such as way, by following the naming convention, enterprises can organize model files in a structure they prefer. In addition, by using the model file name configuration, enterprises can put non-model files together with model files, such as readme and configuration guides when non-model file names have no conflicts with prefixes defined. This can be easily avoided by giving names to non model files starting with a special character, such as an underscore. This approach can effectively organize all model files in a hierarchical structure and also offers the flexibility for enterprises to define a structure based on their own preference.

9.4 **System Architecture**

Software system architecture is closely related to engineering. It provides a blueprint which effectively facilitates software development. System architecture enables complexity and risks to be well managed in the development of enterprise systems. Hence, architecture is a critical factor decisive to the success of software development. Generally, enterprise system architecture should be organized in a way that supports reasoning about the structure, properties and behavior of the system (David Chen, et al. 2008). The roles of architecture in software development include: 1) providing a formal description of a system at a component level; 2) defining the organizational structure of a system and components; 3) identifying interactions and relationships of components; and 4) providing the principles and guidelines governing system design and development. By incorporating these considerations, the architecture is developed as shown in Figure 52. Main advantages of the architecture include: 1) centralized management of semantic models, which motivates model consistency and reuse; 2) multi-layer structure, which organizes functions into different layers for better sharing; and 3) service based information sharing, which ensures information correctness and consistency.

As illustrated in Figure 52, the backend of the system consists of semantic model management, application services and system service. The system services provide functions for security management, system administration and so on. The application services are further separated into three layers: foundation layer, functional layer and domain layer. The foundation layer provides fundamental capabilities to interact with the database and semantic model management service. The functional layer consists of common functions that can be shared by domain services. The domain layer provides business functions to populate and associate information based on business requirements.

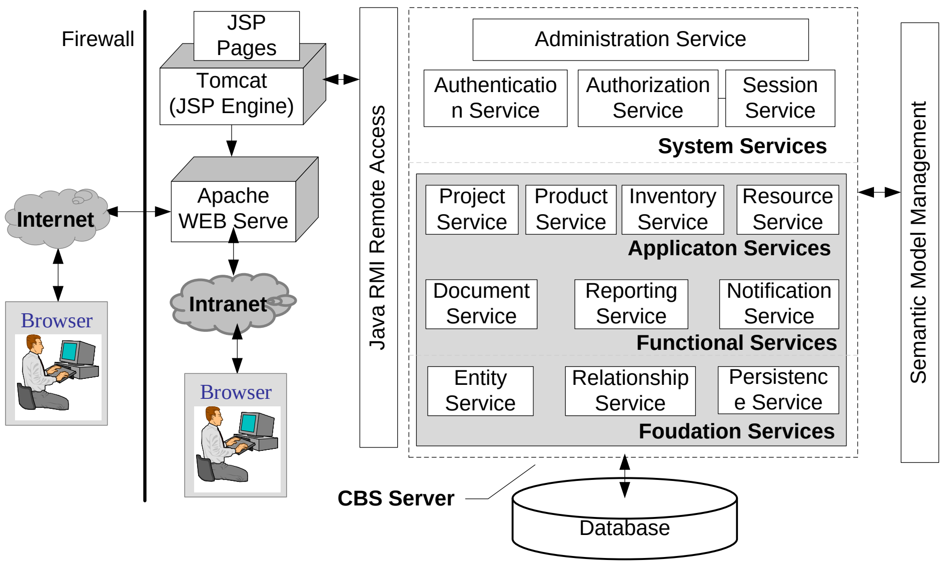


Figure 52 System architecture

On the foundation layer, the entity service manages information entities based on semantic models to support the persistence service. It also has the ability to construct editing models based on semantic models to support the functional service and the domain service. The relationship service provides functions to support relationship management and relationship navigation based on semantic relationship models. It guarantees data integrity and consistency by dynamically interpreting semantic models. The persistence service acts as a gateway to access the database. When storing information entities, it maps information entities to corresponding tables based on semantic table mapping and field mapping. While retrieving information from the database, it maps database records to entity objects.

Built on the top of the foundation layer, the functional layer provides more specific functionalities to support the domain services. The document service manages documents and their ownerships, such as engineering drawings and their relationships with products, parts and designers. One of the main functions of the service is to serialize documents as binary objects and desterilize binary objects as documents to support the persistence service and the relationship service. Reports are managed as a special type of documents. The report service provides functions to generate reports based on semantic models. Generated reports are managed by using the document service.

The functional deployment designed in this architecture enables some of the service components to be easily replaced. It also effectively supports incremental implementation. By clustering functions into different layers and components, the architecture makes the foundation service generic and sharable to different applications. As mentioned above, information sharing between different applications is achieved via the services. Individual applications are motivated to acquire shared information through services rather than direct database access. The significance of this sharing mechanism is that the correctness and consistency of shared information is ensured because the services take care of every aspect of data, such as maturity, validity, integrity, consistency and security.

9.5 Semantic Model Management

In model driven enterprise systems, semantic models need to be well managed and made ready for software programs to use. The semantic model management framework is developed as shown in Figure 53. The semantic models are parsed using SAX (Simple API for XML) APIs in Java. The object mapping bean converts semantic models into Java objects, named model objects, based on the configurable object model shown in Figure 50. Model objects are managed in pool. The model service is an interface for software programs to retrieve model objects.

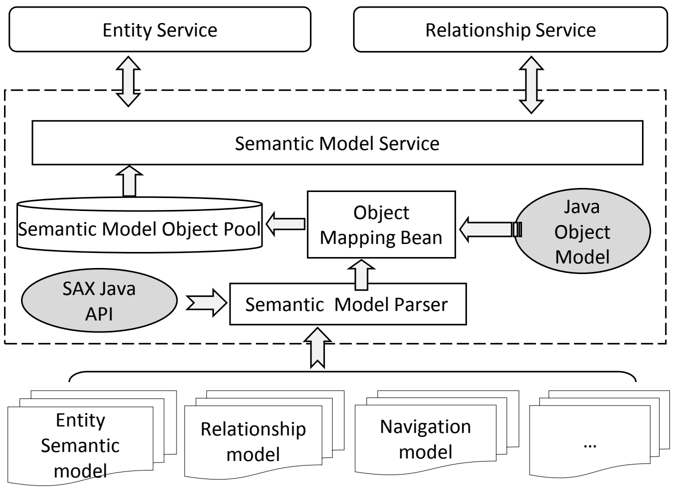


Figure 53 Semantic model management

Figure 54 shows the model of the semantic model management server. The class *Server* represents the semantic model management server. The class *ServerManager* has the capability to manage multiple instances of server. Each server has a context associated, which is represented by the class *CBDContext*. Meanwhile, the class *CBDConext* manages semantic models in a raw format. The raw semantic models are basically XML documents. This enables to develop GUI based tools to manipulate semantic models. The class *CBDContext* reads XML documents using various XML parsers. Services, such as *ObjectConfigService* and *ModuleService*, convert XML documents to semantic model objects based on the object model shown in Figure 50. The semantic model objects are passed to the semantic model service, represented by the class *SemenaticModelService*, and managed in the object pool. The semantic model service provides an interface for the foundation services and other services to retrieve semantic model objects. Figure 55 illustrates the administration screen of the semantic model management server. There are three instances associated with the server manager.

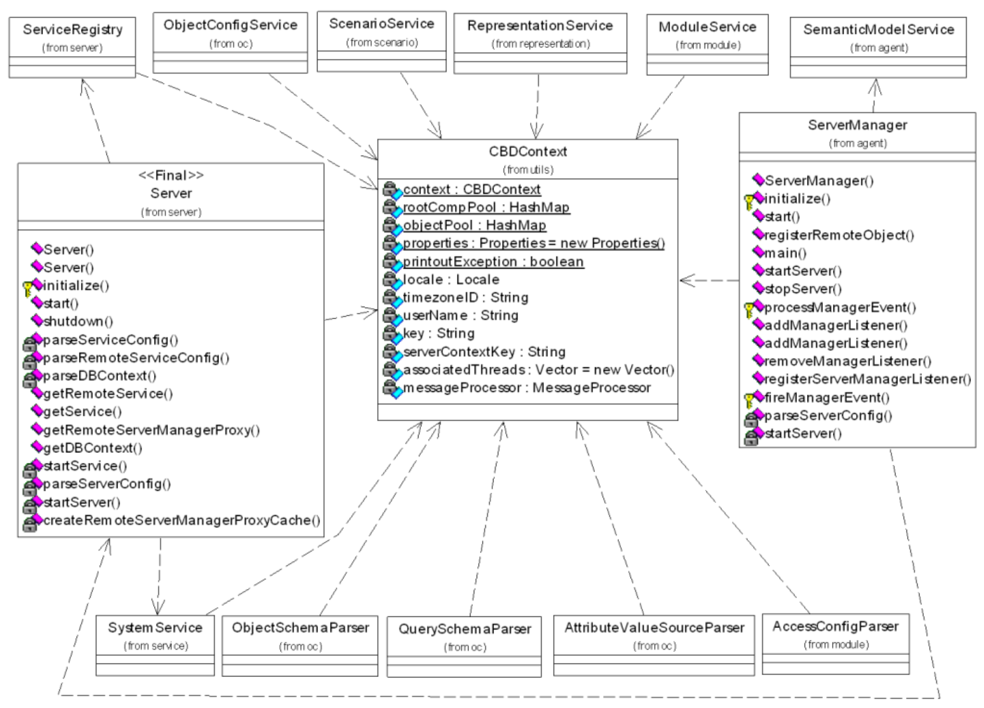
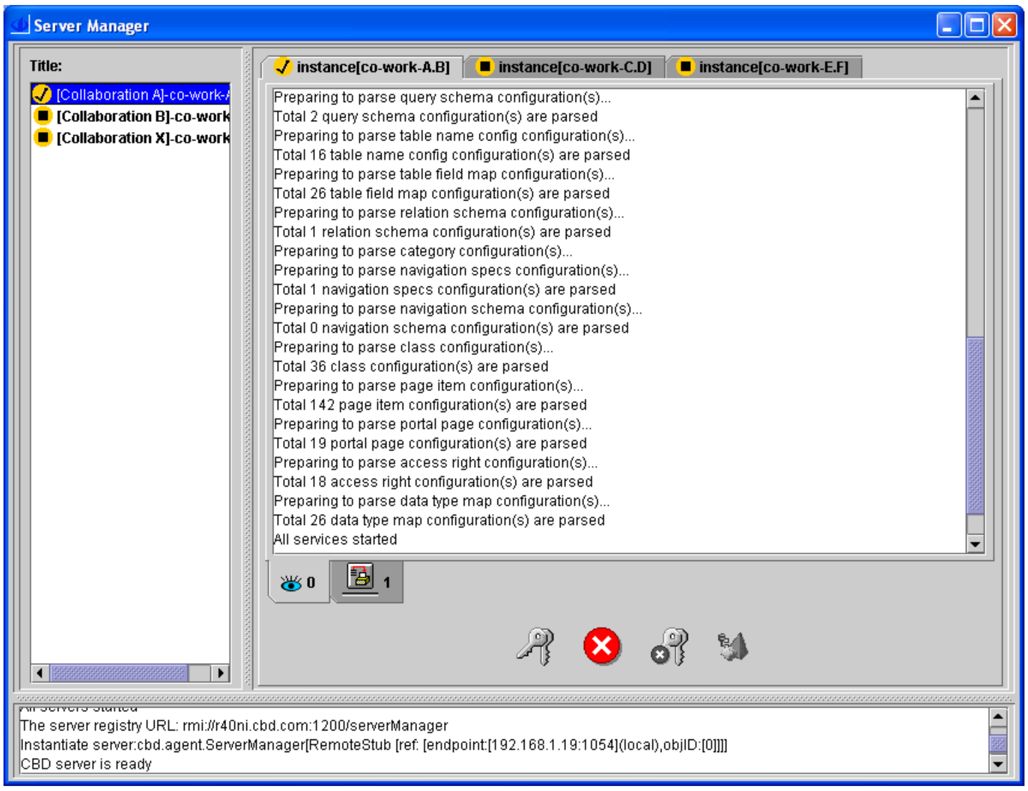

ServiceRegistry (from server)
ObjectConfigService (from oc)
ScenarioService (from scenario)
RepresentationService (from representation)
ModuleService (from module)
SemanticModelService (from agent)
CBDContext (from utils)
context : CBDContext
rootCompPool : HashMap
objectPool : HashMap
properties : Properties = new Properties()
printoutException : boolean
locale : Locale
timezoneID : String
userName : String
key : String
serverContextKey : String
associatedThreads : Vector = new Vector()
messageProcessor : MessageProcessor
<<Final>> Server (from server)
Server()
Server()
initialize()
start()
shutdown()
parseServiceConfig()
parseRemoteServiceConfig()
parseDBContext()
getRemoteService()
getService()
getRemoteServerManagerProxy()
getDBContext()
startService()
parseServerConfig()
startServer()
createRemoteServerManagerProxyCache()
ServerManager (from agent)
ServerManager()
initialize()
start()
registerRemoteObject()
main()
startServer()
stopServer()
processManagerEvent()
addManagerListener()
addManagerListener()
removeManagerListener()
registerServerManagerListener()
fireManagerEvent()
parseServerConfig()
startServer()
SystemService (from service)
ObjectSchemaParser (from oc)
QuerySchemaParser (from oc)
AttributeValueSourceParser (from oc)
AccessConfigParser (from module)

Figure 54 Service model of semantic model management



```
Preparing to parse query schema configuration(s)...
Total 2 query schema configuration(s) are parsed
Preparing to parse table name config configuration(s)...
Total 16 table name config configuration(s) are parsed
Preparing to parse table field map configuration(s)...
Total 26 table field map configuration(s) are parsed
Preparing to parse relation schema configuration(s)...
Total 1 relation schema configuration(s) are parsed
Preparing to parse category configuration(s)...
Preparing to parse navigation specs configuration(s)...
Total 1 navigation specs configuration(s) are parsed
Preparing to parse navigation schema configuration(s)...
Total 0 navigation schema configuration(s) are parsed
Preparing to parse class configuration(s)...
Total 36 class configuration(s) are parsed
Preparing to parse page item configuration(s)...
Total 142 page item configuration(s) are parsed
Preparing to parse portal page configuration(s)...
Total 19 portal page configuration(s) are parsed
Preparing to parse access right configuration(s)...
Total 18 access right configuration(s) are parsed
Preparing to parse data type map configuration(s)...
Total 26 data type map configuration(s) are parsed
All services started
```

```
All servers started
The server registry URL: rmi://r40ni.cbd.com:1200/serverManager
Instantiate server:cbd.agent.ServerManager[RemoteStub [ref: [endpoint:[192.168.1.19:1054](local),objID:[0]]]]
CBD server is ready
```

Figure 55 Screen of model management server

9.6 Expression Model

Expressions are widely used in model driven enterprise systems, such as query model, value validation and reporting. Figure 56 illustrates the expression object model. In the model, the class *ExpressionConstant* contains all reserved keywords, such as function names. The class *ExpressionOperatorTypes* represents operators as objects to facilitate expression evaluation. The class *ExpressionParser* is created to read semantic expressions in XML. The class *ExpressionService* transforms XML based expressions to model objects based on the class *ExpressionConfig*. The semantic management model server further converts the expression model objects to expression trees for other services to use. The expression calculator, represented by the class *ExpressionCalculator*, traverses the entire expression tree to evaluate expressions.

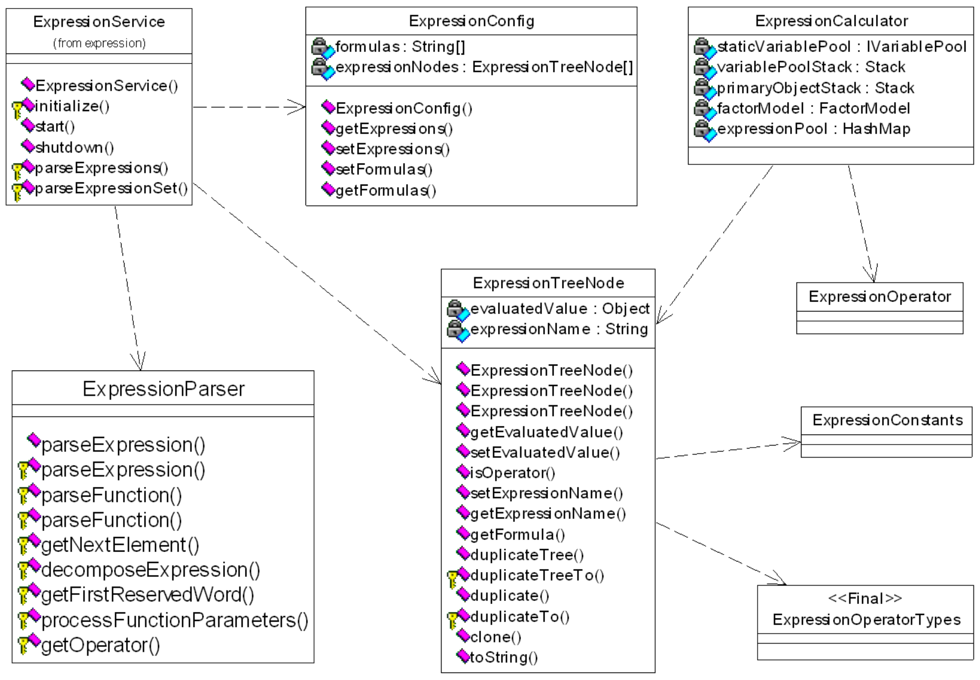
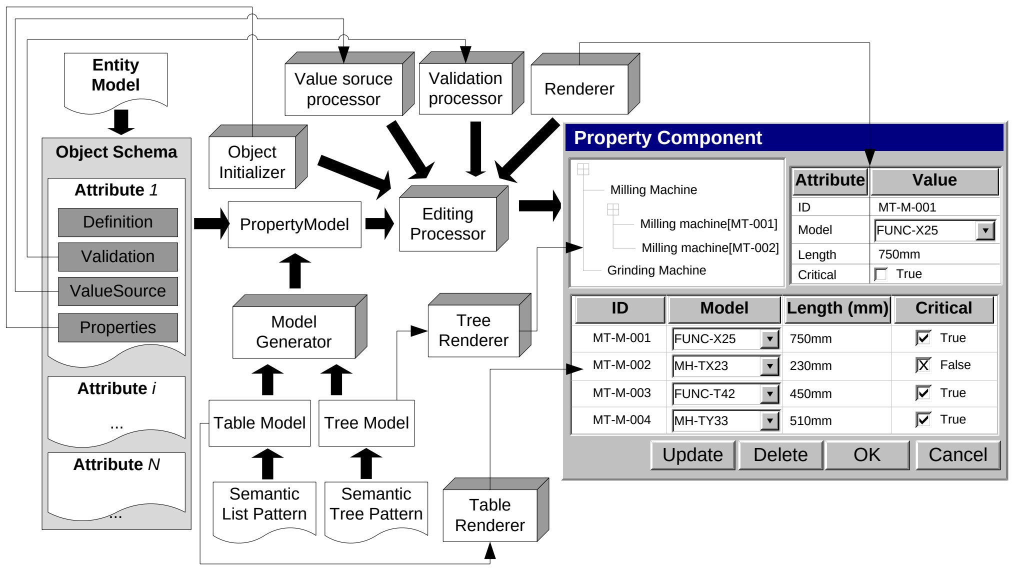


Figure 56 Expression object model

9.7 Entity Management

A framework to manage entities based on semantic models is developed, as shown in Figure 57. This framework can interpret semantic models and manage information entities based on semantic models. It is essential that the entity management can present entity attributes based on semantic models. As shown in the framework, various renderers, such as list renderer, tree renderer and table renderer are modeled to achieve this goal. An adaptive property editing component is developed to automatically present attributes as a 2-column table with suitable editors for end users to view or edit attributes. The object initiator creates and initializes entity instances based on semantic models. The process of initiating an instance is shown in Figure 56.

The object initiator returns an instance of the class *ObjectDataModel* which is a unified data structure to represent entity instances. The object initiator automatically incorporates the corresponding semantic model into the instance of *ObjectDataModel*. The property component is controlled by a property editing model named *PropertyModel*. *PropertyModel* constructs suitable editors for entity attributes according to semantic models. If there is no editing component configured for an attribute, *PropertyModel* constructs a default editor based on the value type and value source of the attribute. *EditingProcessor* is a connector to link the property editing component with the editing property model and other processors. The property editing component notifies the editing processor of value changes. *EditingProcessor* triggers the value validation processor to do validation. After validation, the editing processor notifies the property editing model to update the value back to *ObjectDataModel*. Ultimately, this framework can effectively present an editing environment to manage information entities according to semantic models. *EditingProcesssor* links renderers, value source processor and validation processor to ensure that each attribute has a proper editor and that the attribute value assigned is correct, and value candidates are derived if the value source is configured. To assist information presentations, various GUI components are also developed with the ability to work based on semantic models, such as table, tree and list.



Attribute	Value
ID	MT-M-001
Model	FUNC-X25
Length	750mm
Critical	True

ID	Model	Length (mm)	Critical
MT-M-001	FUNC-X25	750mm	✓ True
MT-M-002	MH-TX23	230mm	False
MT-M-003	FUNC-T42	450mm	✓ True
MT-M-004	MH-TY33	510mm	✓ True

Figure 57 Adaptive entity management environment

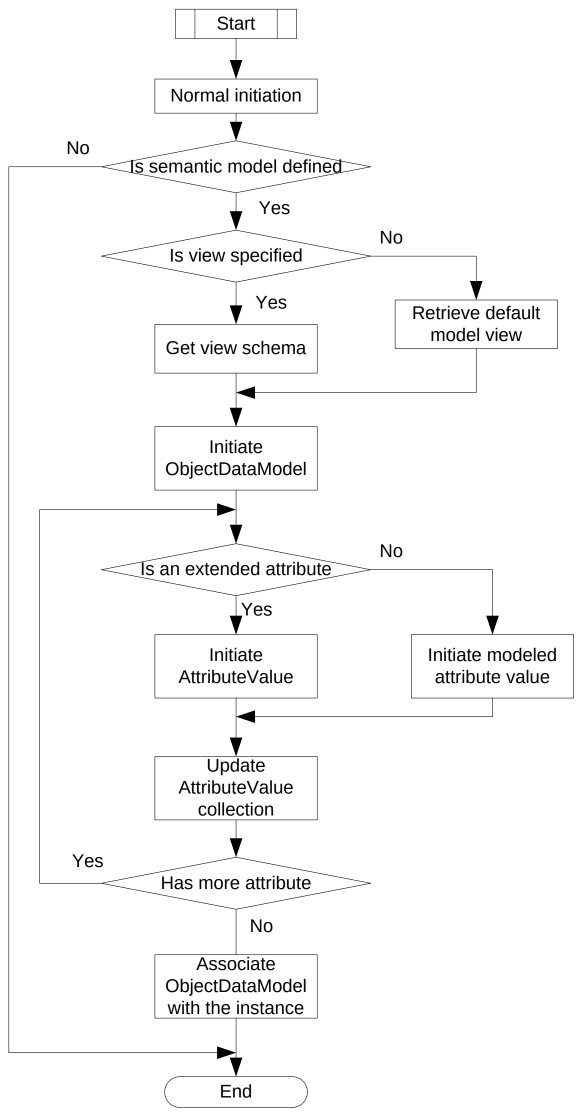


Figure 58 Process of initiating an entity instance

9.8 Relationship Management

Similar to the entity management, an adaptive environment is also needed to manage relationships based on semantic models. In relationship management, relationships are classified into four types based on the cardinality and whether attributes exist in link classes: one-to-one without link attributes, one-to-one with link attributes, one-to-many without association attributes and one-to-many without association attributes. As shown in Figure 59, objects involved in a relationship are presented in a way similar to the entity management. The relationship manager interacts with the relationship service to obtain semantic relationship models. It also constructs various objects involved in a relationship. The relationship management component dynamically presents a management environment by communicating with the relationship manager at runtime. Figure 60 shows an example GUI for managing relationship with properties in the link class.

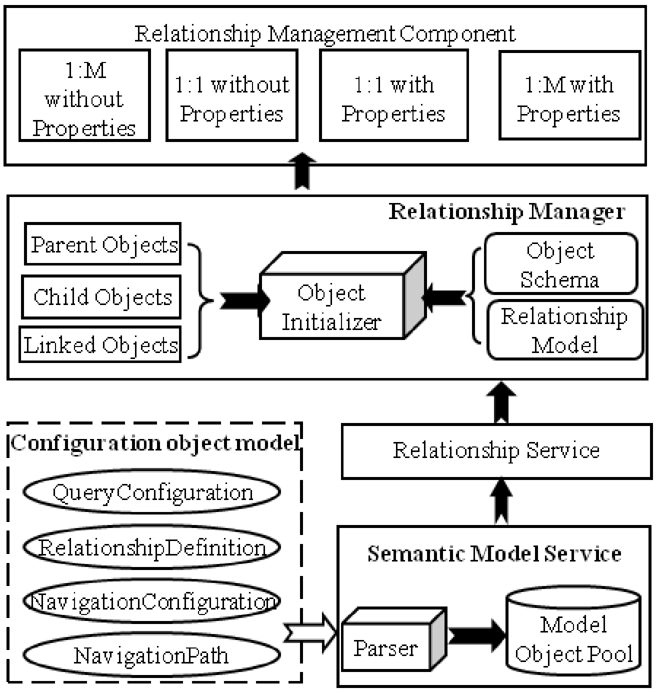
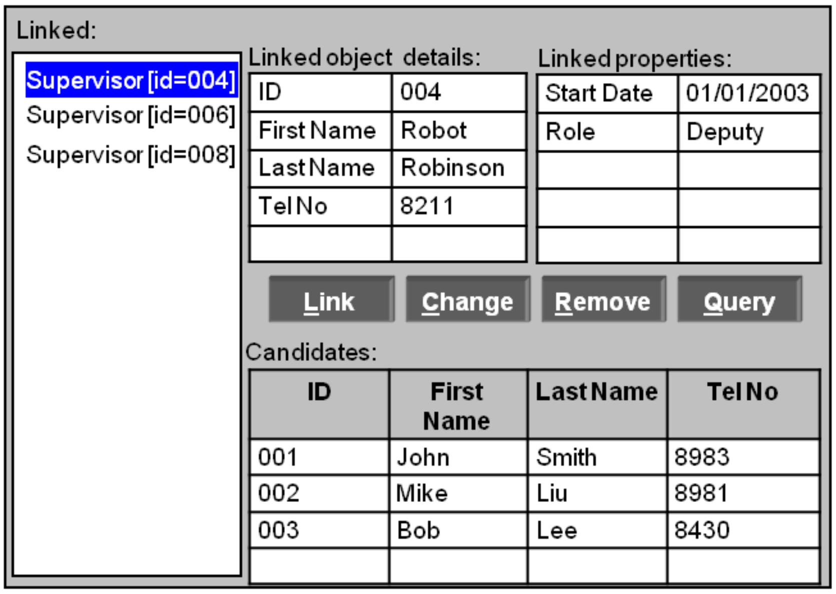


Figure 59 Adaptive relationship management environment



Linked object details:

ID	004
First Name	Robot
Last Name	Robinson
Tel No	8211

Linked properties:

Start Date	01/01/2003
Role	Deputy

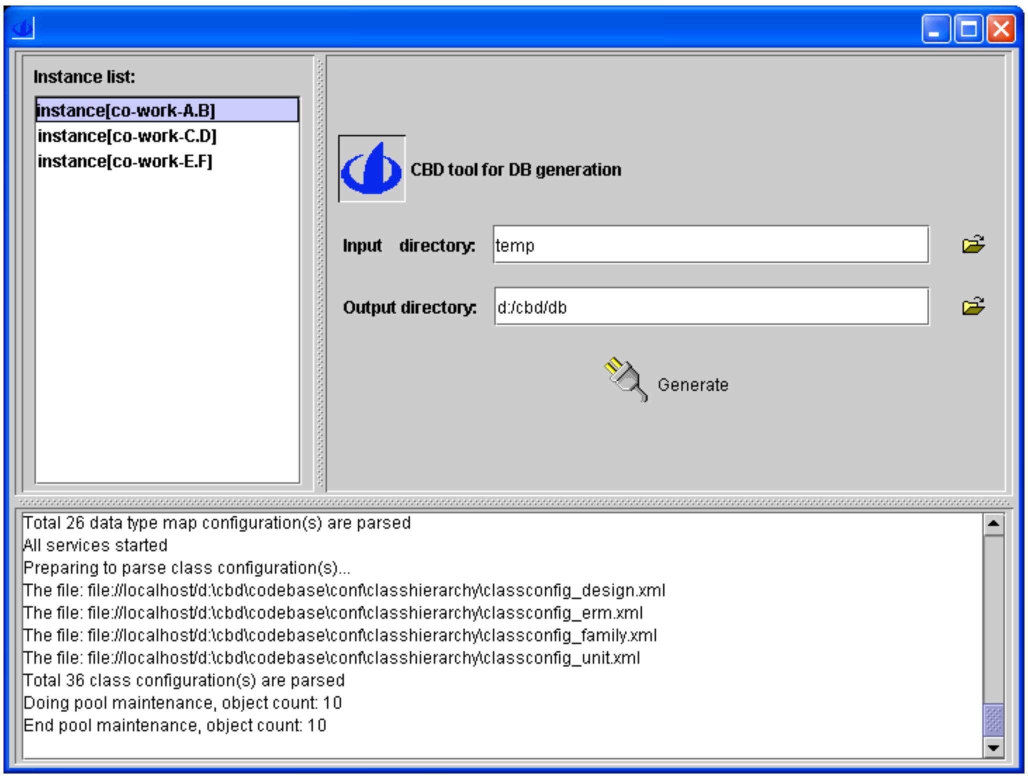
Candidates:

ID	First Name	Last Name	Tel No
001	John	Smith	8983
002	Mike	Liu	8981
003	Bob	Lee	8430

Figure 60 Adaptive relationship management environment

9.9 Database Schema Generation Tool

Semantic models are central resources in the development of model driven enterprise systems. They are not only used to control and guide software programs but also can be used to generate database schema and entity classes. This enables the synchronization of semantic models, platform specific classes, database schema and other resources. The several tools are developed as the part of the prototype system to demonstrate such benefits of model driven enterprise systems. Figure 61 illustrates a tool for generating database schema based on semantic models. Figure 62 shows the database schema generated based on semantic models for product structure management. This tool itself is developed using the model driven approach based on the object model shown Figure 63. Figure 64 is the semantic model to control the GUI layout.



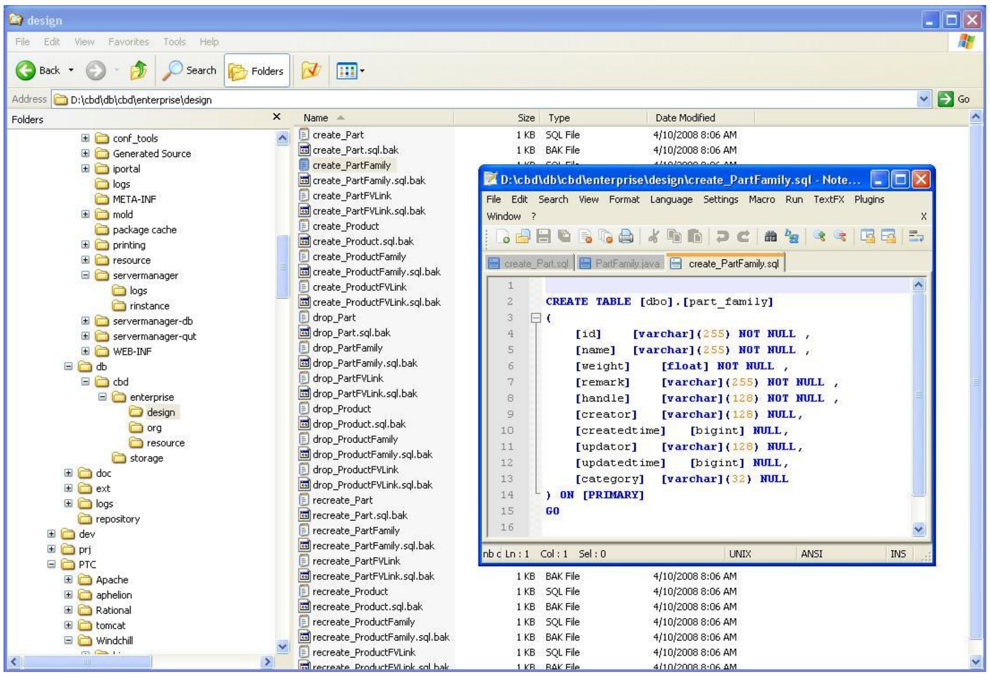
instance[co-work-A.B]
instance[co-work-C.D]
instance[co-work-E.F]

Input directory: temp

Output directory: d:/cbd/db

```
Total 26 data type map configuration(s) are parsed
All services started
Preparing to parse class configuration(s)...
The file: file://localhost/d:\cbd\codebase\conf\classhierarchy\classconfig_design.xml
The file: file://localhost/d:\cbd\codebase\conf\classhierarchy\classconfig_erm.xml
The file: file://localhost/d:\cbd\codebase\conf\classhierarchy\classconfig_family.xml
The file: file://localhost/d:\cbd\codebase\conf\classhierarchy\classconfig_unit.xml
Total 36 class configuration(s) are parsed
Doing pool maintenance, object count: 10
End pool maintenance, object count: 10
```

Figure 61 Database scheme generation tool



```
create_Part                      1 KB  SQL File  4/10/2008 8:06 AM
create_Part.sql.bak              1 KB  BAK File  4/10/2008 8:06 AM
create_PartFamily
create_PartFamily.sql.bak
create_PartFVLink
create_PartFVLink.sql.bak
create_Product
create_Product.sql.bak
create_ProductFamily
create_ProductFamily.sql.bak
create_ProductFVLink
create_ProductFVLink.sql.bak
drop_Part
drop_Part.sql.bak
drop_PartFamily
drop_PartFamily.sql.bak
drop_PartFVLink
drop_PartFVLink.sql.bak
drop_Product
drop_Product.sql.bak
drop_ProductFamily
drop_ProductFamily.sql.bak
drop_ProductFVLink
drop_ProductFVLink.sql.bak
recreate_Part
recreate_Part.sql.bak
recreate_PartFamily
recreate_PartFamily.sql.bak
recreate_PartFVLink
recreate_PartFVLink.sql.bak      1 KB  BAK File  4/10/2008 8:06 AM
recreate_Product                 1 KB  SQL File  4/10/2008 8:06 AM
recreate_Product.sql.bak         1 KB  BAK File  4/10/2008 8:06 AM
recreate_ProductFamily           1 KB  SQL File  4/10/2008 8:06 AM
recreate_ProductFamily.sql.bak   1 KB  BAK File  4/10/2008 8:06 AM
recreate_ProductFVLink           1 KB  SQL File  4/10/2008 8:06 AM
```

D:\cbd\db\cbd\enterprise\design\create_PartFamily.sql - Note...

```

CREATE TABLE [dbo].[part_family]
(
    [id]     [varchar](255) NOT NULL ,
    [name]   [varchar](255) NOT NULL ,
    [weight]     [float] NOT NULL ,
    [remark]     [varchar](255) NOT NULL ,
    [handle]     [varchar](128) NOT NULL ,
    [creator]    [varchar](128) NULL,
    [createdtime]    [bigint] NULL,
    [updator]    [varchar](128) NULL,
    [updatedtime]    [bigint] NULL,
    [category]   [varchar](32) NULL
) ON [PRIMARY]
GO

```

nb c Ln : 1 Col : 1 Sel : 0 UNIX ANSI INS

Figure 62 Generated database schema

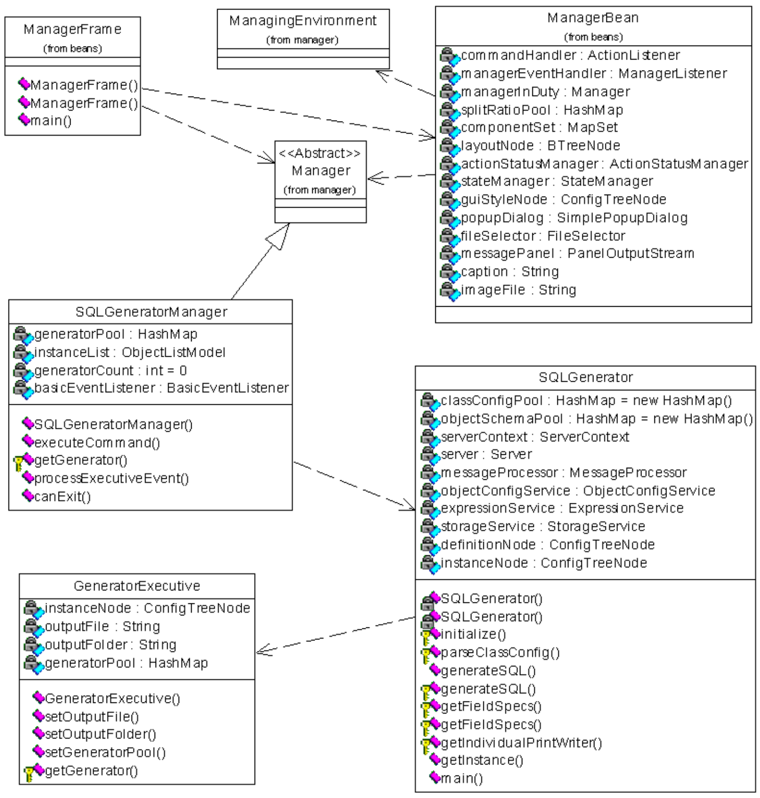


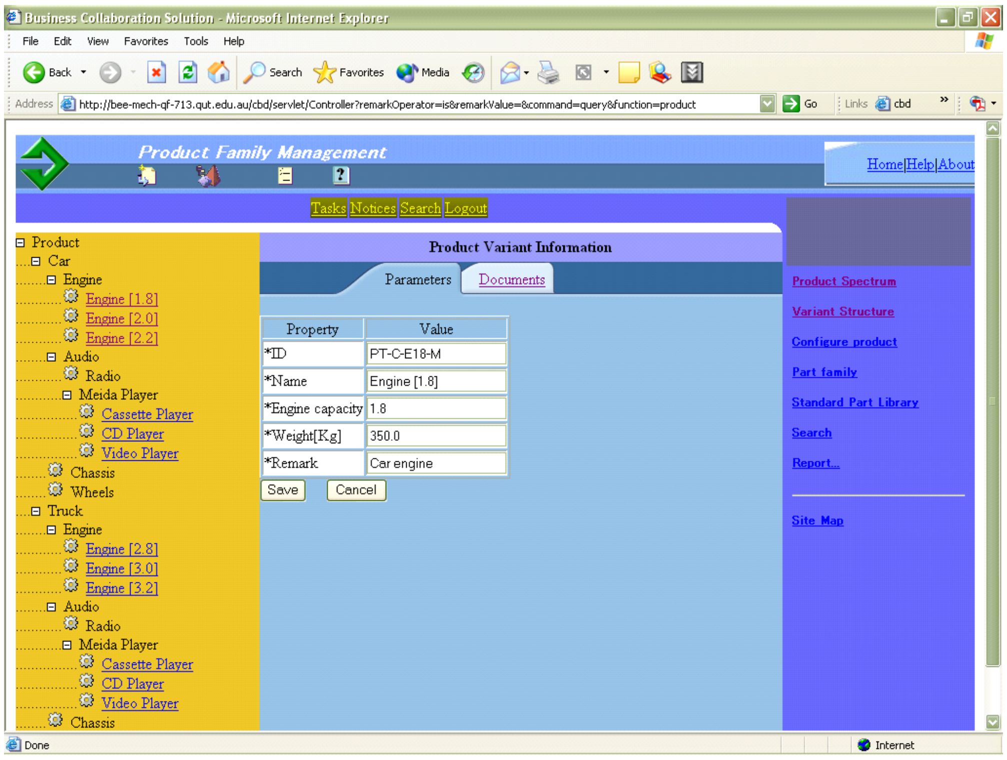
Figure 63 Object model of database schema generation tool

```
<conf type="environment" resourceLink="resource.com.comResource" title="Environment">
        <environment resource="resource.com.comResource">
                <components>
                        <component key="logoLabel">
                        <type>label</type>
                        <image>resource/image/logo.gif</image>
                        <title>_localizedText(DB_GEN_LOGO_LABEL_TITLE)</title>
                </component>
                <component key="outputFile">
                        <action>outputFile</action>
                        <type>button</type>
                        <auxiliaryCommand>selectDirectory</auxiliaryCommand>
                        <propertyName>outputFile</propertyName>
                        <editingModel>default</editingModel>
                        <class>cbd.beans.bbeans.BButton</class>
                        <image>resource/image/folder_o.gif</image>
                        <statusManaged>true</statusManaged>
                        <style>imageButton</style>
                </component>
                <component key="outputDir">
                        <action>outputDir</action>
                        <type>button</type>
                        <auxiliaryCommand>selectDirectory</auxiliaryCommand>
                        <propertyName>outputDir</propertyName>
                        <editingModel>default</editingModel>
                        <class>cbd.beans.bbeans.BButton</class>
                        <image>resource/image/folder_o.gif</image>
                        <statusManaged>true</statusManaged>
                        <style>imageButton</style>
                </component>
                …
                <layout direction="vertical" windowWidth="550" windowHeight="500" >
                        <split key="verticalSplit" direction="vertical" splitRatio="0.5">
                        <componentGroup key="top" direction="horizontal" extending="both" >
                                <split key="horizontalSplit" direction="horizontal" splitRatio="0.3">
                        <componentGroup direction="vertical" extending="both" growingX>
                                <component key="instances">
                                        <extending>both</extending>
                                        <growingX>1.0</growingX>
                                        <growingY>1.0</growingY>
                                </component>
                    </componentGroup>
                    <componentGroup direction="vertical" extending="both">
                <componentGroup key="logoLabel" direction="horizontal" extending=both" >
                        <component key="logoLabel">
                        <extending>both</extending>
                        <growingX>1.0</growingX>
                        <growingY>1.0</growingY>
                </component>
        </componentGroup>
                …
```

Figure 64   Environment model for database schema generation tool

9.10 **Product Structure Management**

Figure 65 shows the product structure management functions implemented based on the architecture developed using the model driven approach. Figure 66 and Figure 67 illustrate the part family semantic model and semantic scenario models developed for this function.


Business Collaboration Solution - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Media
Address
Go
Links
cbd
Product Family Management
Home|Help|About
Tasks Notices Search Logout
Done
Internet

http://bee-mech-qf-713.qut.edu.au/cbd/servlet/Controller?remarkOperator=is&remarkValue=&command=query&function=product

- Product
  - Car
    - Engine
      - Engine [1.8]
      - Engine [2.0]
      - Engine [2.2]
    - Audio
      - Radio
      - Meida Player
        - Cassette Player
        - CD Player
        - Video Player
    - Chassis
    - Wheels
  - Truck
    - Engine
      - Engine [2.8]
      - Engine [3.0]
      - Engine [3.2]
    - Audio
      - Radio
      - Meida Player
        - Cassette Player
        - CD Player
        - Video Player
    - Chassis

Product Variant Information

Parameters | Documents

Property	Value
*ID	PT-C-E18-M
*Name	Engine [1.8]
*Engine capacity	1.8
*Weight[Kg]	350.0
*Remark	Car engine

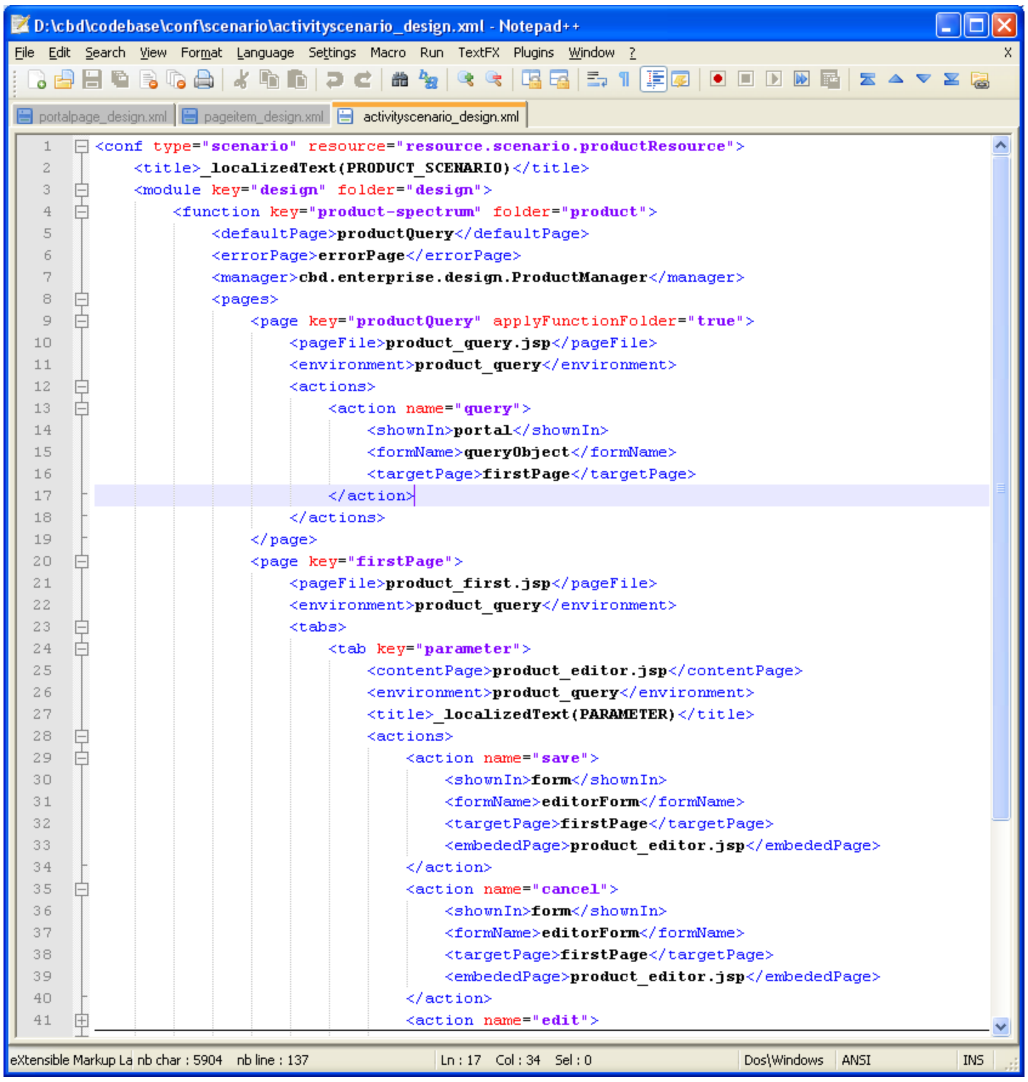
Save   Cancel

- Product Spectrum
- Variant Structure
- Configure product
- Part family
- Standard Part Library
- Search
- Report...
- Site Map

Figure 65   Product structure management

```
<conf type="attributedefinition" resourceLink="attributedefinition_family"
title="attributedefinition_family">
 <object class="cbd.enterprise.family.IFamily">
          <attribute name="id">
          <type>VARCHAR</type>
          <required>true</required>
          <extended>false</extended>
          <title>id.1</title>
          <valueProperties type="text">
                     <defaultValue>xxx</defaultValue>
                     <compulsory/>
                     <caseMode>none</caseMode>
                     <spacePermitted>false</spacePermitted>
                     <maximalLength>128</maximalLength>
          </valueProperties>
  </attribute>
          <attribute name="name">
          <type>VARCHAR</type>
          <extended>false</extended>
          <title>name.1</title>
          <valueProperties type="text">
                     <caseMode>none</caseMode>
                     <spacePermitted>false</spacePermitted>
                     <maximalLength>128</maximalLength>
          </valueProperties>
  </attribute>
 </object>
  ...
 </object>
</conf>
```

Figure 66   Part family semantic model


D:\cbd\codebase\conf\scenario\activityscenario_design.xml - Notepad++
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
portalpage_design.xml
pageitem_design.xml
activityscenario_design.xml
eXtensible Markup La nb char : 5904 nb line : 137 Ln : 17 Col : 34 Sel : 0 Dos\Windows ANSI INS

```xml
<conf type="scenario" resource="resource.scenario.productResource">
    <title>_localizedText(PRODUCT_SCENARIO)</title>
    <module key="design" folder="design">
        <function key="product-spectrum" folder="product">
            <defaultPage>productQuery</defaultPage>
            <errorPage>errorPage</errorPage>
            <manager>cbd.enterprise.design.ProductManager</manager>
            <pages>
                <page key="productQuery" applyFunctionFolder="true">
                    <pageFile>product_query.jsp</pageFile>
                    <environment>product_query</environment>
                    <actions>
                        <action name="query">
                            <shownIn>portal</shownIn>
                            <formName>queryObject</formName>
                            <targetPage>firstPage</targetPage>
                        </action>
                    </actions>
                </page>
                <page key="firstPage">
                    <pageFile>product_first.jsp</pageFile>
                    <environment>product_query</environment>
                    <tabs>
                        <tab key="parameter">
                            <contentPage>product_editor.jsp</contentPage>
                            <environment>product_query</environment>
                            <title>_localizedText(PARAMETER)</title>
                            <actions>
                                <action name="save">
                                    <shownIn>form</shownIn>
                                    <formName>editorForm</formName>
                                    <targetPage>firstPage</targetPage>
                                    <embededPage>product_editor.jsp</embededPage>
                                </action>
                                <action name="cancel">
                                    <shownIn>form</shownIn>
                                    <formName>editorForm</formName>
                                    <targetPage>firstPage</targetPage>
                                    <embededPage>product_editor.jsp</embededPage>
                                </action>
                                <action name="edit">
```

Figure 67 Semantic scenario model of product structure management

9.11 Conclusions

This chapter briefly demonstrated a proof-of-concept system, key components and a database scheme generation tool developed based on the concept of model driven enterprise systems and semantic representations. It proves that semantic models can be effectively interpreted by computers to control software programs. The system architecture developed successfully separates semantic models and software programs. It enables the semantic model service to manage semantic models well, at a centralized place for applications to use. This makes semantic models sharable to various applications.

9.11.1 Challenges in Development of Model Driven Enterprise System

Through the prototype development, the following two challenges can be identified in the development of model driven enterprise systems:

- In the design phase, additional efforts are needed to investigate the variability of business requirements and design semantic models to support the variability. Since model driven enterprise systems do not work directly based on particular business requirements, an extra level abstraction is essential by transforming specific business requirements to more general model instructions at the design. The degree of enterprise system flexibility is tightly dependent on the level of the abstraction. Two approaches can be adopted to achieve a better abstraction: 1) collect and analyze business requirements from different companies, better from different industrial sectors. This facilitates the design of semantic representations that can support more variations; 2) study existing systems. This approach can help design powerful semantic representations to achieve more flexible GUI environments, information presentation and functional layout.
- In the test phase, in addition to normal functions tests based on business requirements, the ability of software programs to support different variations of a semantic model also needs to be thoroughly tested. This type of test can be very difficult because a semantic model can be varied in many ways. Two approaches can be adopted to ease this type of test. One is to use composite semantic models to challenge relevant software programs. A composite semantic model is a hypothetic semantic model which mimics the most complicated case of a semantic model. Another is to use a test-driven development approach. The test driven development approach has two purposes: 1) to ensure that software programs work as assumed at the development stage by mocking up different inputs; and 2) to check that code changes do not break exiting functions. By adopting the test driven development approach, various variations of a semantic model

can be mocked up to test relevant software programs to achieve a higher level of stability.

9.11.2 Advantages of Semantic Representations

The prototype proves that semantic representations are very effective to support the development of model driven enterprise systems. Semantic representations provide two main advantages in addition to controlling the behavior of software programs:

- Firstly, semantic models enable processing logic that needs to repeat again and again to be unified. Taking reporting as an example, the processes to create different reports are very similar. However, the piece of code for creating one type of report can be not reused for another type of report because information and report appearances are different. In this situation, similar processes have to be implemented using different pieces of code many times. It incurs a long development time and makes maintenance difficult. By introducing a set of semantic models, different pieces of code for different types of reports can be unified.
- Secondly, semantic representations can be used to develop generic GUI components. For example, in enterprise systems, a common way to present multiple entities for end users to select is to list a set of entities as a table. Though there are some table components which are very generic, different pieces of code are still needed when presenting different entities due to the difference in attributes and table heads etc. By using semantic representation, table heads and attributes to be displayed and can be defined in a semantic model. In addition, attribute specifications are available from semantic attribute definitions. Mapping models and pattern models can be integrated to transform attribute values for display. As a result, a very generic table component can be developed. When displaying a set of entities, all the work that needs to be done is to pass the entities to table and specify a semantic model. Such a table component can be used to present any types of objects.

As a whole, semantic representations can be effectively used to unify similar processing logic and generalize GUI components. Less code means less development time and easier maintenance. The prototype development reveals that semantic representations can be incrementally introduced to a software system. A consequence is that semantic representations can be used to re-engineer enterprise systems to incrementally increase their flexibility.

9.11.3 Limitations of Prototype

The primary objective of this prototype is to investigate a way to integrate semantic models with business object models and a systematic approach to organize semantic models for software programs to use. Due to the time constraint, the following things have not been touched:

- Generating semantic models based on models established using other tools, such as UML tools;
- A collection of semantic models needs to be constructed based on another business domain to challenge the software programs developed in this prototype;
- A fundamental guideline needs to be developed to assist the transform of specific business requirements to generic semantic models;
- Various common semantic models can be identified, especially for GUI components, such as list, tree, and table.

Chapter 10 Conclusions and Future Research

10.1 **Research Summary**

In the 1980s, enterprises started reforming their organizational structure, re-engineering their business process and adopting information technology for the flexibility to rapidly respond to internal and external changes. Throughout this process, computers in enterprises have evolved from standalone facilities to complicated, interconnected network systems. The goal of software applications shifts, from assisting individuals, to connecting various functional units. The enterprise itself transforms from relatively independent departments to an interdependent environment. As a result, enterprise systems have been tightly coupled with business operation. Business flexibility requires the support of enterprise systems. Due to the lack of flexibility, enterprise systems cannot be changed rapidly to catch up with business changes and often drag business behind. Individualization of an enterprise system is resource intensive. Much effort is needed to redesign and redevelop functions for specific needs. Flexible enterprise systems are strongly desired to ease ongoing business needs quickly and effectively. Enterprise system vendors are confronting the challenge to deliver flexible enterprise systems. However, research on systematic methods for developing flexible enterprise systems has not received enough attention.

Extensive literature has addressed this issue by identifying success or failure factors, implementation approaches, and project management strategies. Those efforts were aimed at learning lessons from post implementation experiences to help future projects. This research looked into this issue from a different angle. It addressed this issue by delivering a systematic method for developing flexible enterprise systems which can be easily tailored for different business practices or rapidly adapted when business practices change.

Chapter 3 initiated the concept of model-driven enterprise systems by leveraging the convergence of MDA and workflow management. The novelty of the concept is to separate system models from software programs. In such a system, models act as instructors to guide and control software programs. Software programs play the role of executives in completing processing functions according to instructions in models. Since semantic models stay outside of programs, semantic models can be changed or replaced. After models are changed, programs can behave in a different way. This concept offers the opportunity to tailor enterprise systems by reconstructing system models.

Based on the initiated concept, Chapter 4 identified various types of system models that need to be extracted from software programs. These models need to be represented in a language which can be easily understood and modified by human beings and can also be effectively interpreted by computers. Various types of semantic representations were investigated for constructing these system models.

To verify the concept and semantic representations, Chapter 5 developed a comprehensive business process model based on the general practice of the manufacturing industry. Based on this business process model, resource management, product structure management and reporting are selected as study cases.

Chapter 6, Chapter 7 and Chapter 8 developed a semantic resource model, semantic product structure model and semantic reporting as case studies. These case studies proved that semantic representations can be used to represent complex business entities, relationships and business logic. Chapter 9 integrated business object models developed in the case studies and developed a proof-of-concept prototype system based on the concept of model-driven enterprise systems and semantic representations. Lessons learnt from the prototype development were discussed.

10.2 **Research Contributions**

10.2.1 Concept of Model Driven Enterprise Systems

The concept of model driven enterprise systems is the primary contribution of this research. Business requirements, design decisions and developers' thinking are usually hard coded into enterprise systems throughout the development process. In such a way, enterprise system models dissolve into software programs and cannot stand independent of software programs. After a system is developed, system models become intangible. This eliminates the possibility of adjusting enterprise systems by changing system models. Changes to system models need to be implemented by revising system source code. The concept of model driven enterprise systems provides a novel paradigm to separate system models from software programs and enables system models to stay outside of software programs. The separation of system models from software programs exposes an opportunity to mediate the behavior of enterprise systems through modifying system models. This is critical to flexible enterprise systems.

10.2.2 Semantic Representations

Semantic representations are also a significant contribution of this research. Traditionally, system models exist dependent of software programs. They are reflected in system source code. Model driven enterprise systems require system models to be extracted from software programs. This research has identified various types of system models that need be extracted from software programs by developing an abstraction model of enterprise systems. Then, various semantic representations are developed, including semantic entity representation, entity relationship representation, business logic representation, function layout representation and GUI environment representations. With these semantic representations, entity models and relationship models can be declared outside of software programs; and business logic, function layout logic and information presentations logic can be represented, described as semantic models. These system models stay independent of, and loosely coupled with software programs. They can be easily constructed by human beings. At the same time, they can also be effectively interpreted by computers.

10.2.3 Promoting Role of System Models

Another key contribution of this research is the promotion of the role of system models from guiding writing system source code to controlling the behavior of enterprise systems. The software development lifecycle (SDLC) provides a philosophy to manage the process of enterprise system development. In SDLC, business requirement collection, system design and system coding are major steps to ensure that an enterprise system is developed in line with business requirements. Traditionally, software developers write system source code by understanding business requirements and design decisions. After being developed, an enterprise system works by following the way developers have defined. In other words, developers' thinking is implanted into enterprise systems. Developers' thinking is the understanding of business requirements and design decisions. Business requirements and design decisions are usually represented as various system models. Consequently, the major role system models play is to guide developers to write system source code.

10.2.4 Advancement to MDA

The concept of model-driven enterprise systems moves MDA a big step forward. From the model usage perspective, MDA uses models to generate platform specific code. The key contribution MDA made to the area of enterprise system development is the establishment of a direct connection between system models and system source code. The concept of model-driven enterprise systems advances MDA and promotes system models from guiding writing system source code to controlling the behaviors of enterprise systems. From the model existence perspective, system models in MDA dissolve into system source code. System models stand independent of software programs. Graphical models are still there after an enterprise system is developed but they are same as other documents. They don't have a direct connection to software programs. In model-driven enterprise systems, system models are loosely coupled with software programs to control and guide the execution of software programs. They become a part of an enterprise system. They are still tangible and can be modified to mediate the behaviors of software programs. From the time perspective, system source code in MDA reflects a snapshot of system models. System source code can be synchronized with system models through running system generation tools after system models change. In model-driven enterprise systems, software programs behave differently when models are changed. Software programs can dynamically reflect changes to system models. From the evolution paths of the system model role in enterprise systems, it can be observed that the concept of model-driven, enterprise systems advances is a big advancement of MDA:

- CASE prompted model-driven analysis. In CASE, system models are mainly used for analysis and design decision making. No linkage between system models and system code exists. System models exist as a type of unstructured documents;
- MDA promoted model-driven development. System models are used for system analysis and code generation. System source code is a snapshot of system models. A static linkage is established between system models and system source code. System models exist as a type of structured document;
- Model-driven enterprise systems promote system models to driven software programs. A dynamic linkage is established between system models and software programs. System models stay independent of, and loosely coupled with software programs. System models are a part of enterprise systems.

10.2.5 Enlarged Space for Enterprise System Flexibility

Traditionally, enterprise systems work by following developers' thinking. Developers' thinking is the understanding of business requirements. Therefore, business requirements are hard coded into enterprise systems. Though various parameters can be introduced for adjusting the behavior of enterprise systems, all options have to be predefined. In nature, parameter-based configuration is to choose one of the predefined options. In the paradigm of model driven enterprise systems, designers and developers are motivated to support more model instructions. More model instructions imply to a larger degree, that the enterprise system can vary. Such an enterprise system can accommodate more changes to business practice changes. Consequently, high flexibility can be achieved.

10.3 **Industrial Benefits**

10.3.1 New Approach to System Implementation and Maintenance

A predominant approach of enterprise system implementation is customization which is to redesign and redevelop functions to meet specific requirements. This is a long-cycle process which involves different teams, vendors and third parties. The process of implementing model-driven enterprise systems is mainly the iteration of requirements analysis and model configuration/reconfiguration. Fewer chances exist for developers to be involved because chances to change system source code are very limited. At the system maintenance stage, to support ongoing business changes, the primary work is to reconfigure system models. This is a significant advantage of model-driven enterprise systems. Compared to current parameter based configuration, model-driven enterprise systems provide a semantic and much more intuitive context for constructing system models. Simplified implementation and maintenance process and fewer teams imply lower cost and shorter cycle.

10.3.2 New Approach for Evaluating Enterprise Systems

The selection of enterprise systems needs to be carried out with extensive review and evaluation. After a decision is made, the enterprise is supposed to couple with the selected vendor for a long time. Some reports can be found in literature that enterprise systems are implemented but original expectations could not be achieved. The primary reason is that selected systems do not match business requirements well. These enterprises either invest more to further customize the selected system or simply accept the failure. Model-driven enterprise systems can be effectively evaluated by using system models to check how specific requirements can be satisfied. In the evaluation of model-driven enterprise systems, some preliminary models can be constructed based on special requirements. The result can be seen as soon as the models are incorporated. This enables enterprises to thoroughly evaluate an enterprise system before making a decision. Traditional enterprise systems do not provide this opportunity. To truly evaluate special requirements, system source code needs to be changed. This is not possible in most situations. The achievability of special requirements can only be estimated by evaluating the development toolkit, supported language and exposed APIs by the systems.

10.3.3 Long Enterprise System Life

Because business environments and customer demands keep changing, enterprises often have to change their business practices. Rigid enterprise systems are often quickly phased out of production because they cannot support new business practices. Since model driven enterprise systems provide higher flexibility, this means that such enterprise systems can accommodate more changes. Therefore, these systems can be used for a longer time and dramatically reduce business running cost.

10.4 **Future Research**

This research initiated a concept of enterprise systems and then primarily concentrated on developing semantic representations for system models. Semantic representations are a key technique to enable model-driven enterprise systems. Three main areas can be identified for future research. The outcome of this research has great potential for developing configurable software components for enterprise systems based on semantic models. It can also be adopted by enterprise vendors to development semantic model based enterprise systems. To maximize the commercial potential of the proposed method, further research in the following area may need to be carried.

The first area is to integrate semantic representations with UML. UML is a standardized general-purpose modeling language in the area of software engineering. Various UML compliant tools have been developed for modeling enterprise systems and business processes. As a notation based graphical language, UML standardizes notations but not the format of electronic UML files. Each UML tool has its own proprietary electronic format. Both semantic representations developed in this research and UML are intended to describe the same thing, but for different purposes. Potential is obvious in that semantic representations can be integrated with UML tools. Semantic representations can be promoted as one of standard output formats of UML tools. Many UML tools offer an open interface for add-ons. A simple approach to integrate semantic representations with UML tools is to develop add-ons for existing UML tools. Integration with UML enables existing UML models to be reused to generate semantic representations. This can speed up the construction of the semantic model in the development of model driven enterprise systems. Such integration streamlines the development process of model driven enterprise systems.

The second is to standardize semantic representations. Software development is often compared with hardware development. In hardware development, there has been much progress. For example, processor speed has grown exponentially in the past twenty years. Hardware component replacement is a simple plug-and-play process. However, this is not a case of upgrading and changing software systems. High exchangeability in hardware comes from standardization. Compared to hardware, standardization of software systems and components is far behind. Currently, two types of standards exist in the software industry: API standards and information exchange standards. API standards provide exchangeability within the same platform. For example, the J2EE standard enables enterprises to change their J2EE container from one vendor to another. However, the candidates of containers have to be implemented in Java. One of the most famous standards for information exchange is Web Services. Currently, little possibility exists for establishing standards for the entire enterprise system. Semantic representations enable system models to stay outside of software programs. This offers an opportunity to establish standardized semantic representations for system models. A practical way to do it is to separate system models into multiple levels so that standardization can be done incrementally for each level. For example, at the GUI level, a standard can be established for developing GUI components that can work based on semantic models. From the entity representation perspective, XML tags and keywords can be standardized for declaring attributes, entities and relationships.

The third area is to identify common processing logic and develop semantic representations based on real industrial cases. This research has mainly investigated mapping, pattern and expressions. These three types of logic representations were widely used in various semantic representations for different purposes, such as validation, query filtering and value transforming. Further logic representations can be identified and developed to facilitate the development of model driven enterprise systems.

Bibliography

Aagedal, J. Ø., Bézivin, J., and Linington, P. F. (2005), *Model-Driven Development (Wmdd 2004)* (3344 ed.),

Aalst, W. M. P. v. d. (2002), *Making Work Flow: On the Application of Petri Nets to Business Process Management* (2360 ed.),

Abdmouleh, A., and Spandoni, M. (2004), "Distributed Client/Server Architecture for Cimosa-Based Enterprise Components," *Computers in Industry*, 55, 239-253.

Abeysinghe, G., and Phalp, K. (1997), "Combining Process Modelling Methods," *Information and Software Technology*, 39, 107-124.

Agerfalk, P. J., "Towards Structured Flexibility in Information Systems Development: Devising a Method for Method Configuration," *Journal of database management*, 20, 51.

Agostini, A., and Michelis, G. D. (2000), "Improving Flexibility of Workflow Management Systems," *Business Process Management*, 1806.

Allen, B. R., and Boynton, A. (1991), "Information Architecture: In Search of Efficient Flexibility.," *MIS Quarterly*.

Anonymous. (1999), "Survey: Business and the Internet: Erp Rip?," *The Economist*, 351.

Anonymous. (2004), "Enterprise Workflow," *International Journal of Productivity and Performance Management*, 53, 561.

Applegate, L. M., McFarlan, F. W., and McKenney, J. L. (1999), *Corporate Information Systems Management - Text and Cases, 5th Edition*, McGraw Hill.

Baker, R. P., and Maropoulos, P. G. (2000), "An Architecture for the Vertical Integration of Tooling Considerations from Design to Process Planning," *Robotics and Computer-Integrated Manufacturing*, 16, 121-131.

Barry, J., et al. (1998), "Niiip-Smart: An Investigation of Distributed Object Approaches to Support Mes Development and Deployment in a Virtual Enterprise," in *Enterprise Distributed Object Computing Workshop, 1998. EDOC '98. Proceedings. Second International*, pp. 366-377.

Bauer, B., Müller, J. P., and Roser, S. (2004), *A Model-Driven Approach to Designing Cross-Enterprise Business Processes* (3292 ed.),

Bendoly, E., and Kaefer, F. (2004), "Business Technology Complementarities:
Impacts of the Presence and Strategic Timing of Erp on
B2b E-Commerce Technology Efficiencies," *Omega*, 32, 395–405.

Bernus, P., and Nemes, L. (1997), "Requirements of the Generic Enterprise Reference Architecture and Methodology," *Annual Reviews in Control*, 21, 125-136.

Bieberstein, N., Bose, S., Fiammante, M., Jones, K., and Shah, R. (2005), *Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap* Prentice Hall PTR.

Boelcke, A. (2003), "What Is Analogical Reasoning," *http://www.wisegeek.com/what-is-analogical-reasoning.htm*.

Botta-Genoulaz, V., Millet, P.-A., and Grabot, B. (2005), "A Survey on the Recent Research Literature on Erp Systems," *Computers in Industry*, 56, 510-522

Bradford, M., and Florin, J. (2003), "Examining the Role of Innovation Diffusion Factors on the Implementation Success of Enterprise Resource Planning Systems, International Journal of Accounting," *Information Systems*, 4 205–225.

Brière-Côté, A., Rivest, L., and Desrochers, A. (2010), "Adaptive Generic Product Structure Modelling for Design Reuse in Engineer-to-Order Products," *Computers in Industry*, 61, 53-65.

Brill, P. H., and M, M. (1990), "Measurement of Adaptivity and Flexibility in Production Systems," *European Journal of Operational Research*, 49, 325-332.

Brown, A. (2004), "An Introduction to Model Driven Architecture," *http://www.ibm.com/developerworks/rational/library/3100.html*.

Brown, A. W. (2000), *Large-Scale, Component-Based Development*, Prentice-Hall.

Browne, J., Dubois, D., Rathmill, K., Sethi, S. P., and Stecke, K. E. (1984), "Classification of Flexible Manufactunng System, ," *The FMS Magazine*.

Calisir, F. (2004), "The Relation of Interface Usability Characteristics, Perceived Usefulness, and Perceived Ease of Use to End-User Satisfaction with Enterprise Resource Planning (Erp) Systems, ," *Computers in Human Behavior*, 20, 505–515.

Camarinha-Matos, L. M., and Afsarmanesh, H. (1999a), *Infrastructures for Virtual Enterprises—Networking Industrial Enterprises*, Kluwer Academic Publishers.

Camarinha-Matos, L. M., and Afsarmanesh, H. (1999b), "Tendencies and General Requirements for Virtual Enterprises," *Proceedings of the IFIP TC5 WG5.3 / PRODNET Working Conference on Infrastructures for Virtual Enterprises: Networking Industrial Enterprises*, 153, 15-30.

Carlsson, B. (1989), "Flexibility and the Theory of the Firm," *International Journal of Industrial Organization*, 7, 179-203.

Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1998), "Workflow Evolution," *Data & Knowledge Engineering*, 24, 211-238.

CBDi Forum. (2001), "Application Integration," *CBDi Forum*.

Chalmeta, R., Campos, C., and Grangel, R. (2001), "References Architectures for Enterprise Integration," *Journal of Systems and Software*, 57, 175-191.

Chalmeta, R., and Grangel, R. (2003), "Ardin Extension for Virtual Enterprise Integration," *Journal of Systems and Software*, 67, 141-152.

Chambers, S. (1992), *Flexibility in the Context of Manufacturing Strategy*
ed. C. A. Voss, Chapman & Hall, London.

Chen, R.-S., Sun, C.-M., and Jih, W.-J. (2009), "Factors Influencing Information System Flexibility: An Interpretive Flexibility Perspective," *International Journal of Enterprise Information SystemsPerspective*, 5.

Cheng, J. M. J., Simmons, J. E. L., and Ritchie, J. M. (1997), "Manufacturing System Flexibility: The "Capability and Capacity" Approach," *Integrated Manufacturing Systems*, 8, 147-158.

Chung, C. H., and Chen, I. J. (1990), *Managing Flexibility of Flexible Manufacturing Systems for Competitive Edge*, ed. M. J. Liberatore, Springer, Berlin.

Clive, F., and Aiken, P. H. (1999), *Building Cooperate Portals with Xml*, USA: McGraw-Hill Companies, Inc.

Correa, H., and Slack, N. (1996), "Framework to Analyze Flexibility and Unplanned Change in Manufacturing Systems," *Computer Integrated Manufacturing Systems*
9.

Cowley, S. (2010), "Study: Bpm Market Primed for Growth," http://www.infoworld.com.

CTRC. (1999), "Enterprise Resource Planning: Integrating Applications and Business Process across the Enterprise," *Computer technology Research Corporation, USA*.

Curtis, B., Kellner, M. I., and Over, J. (1992), "Process Modeling," *Communications of the ACM*, 35, 75.

D'Souza, D. E., and Williams, F. P. (2000), "Toward a Taxonomy of Manufacturing Flexibility Dimensions," *Journal of OperationsM anagement*, I8, 577-593.

Das, T. K., and Elango, B. (1995), "Managing Strategic Flexibility: Key to Effective Performance," *Journal of General Management*, 20, 60-75.

Davenport, T. (1998), "Putting the Enterprise into the Enterprise System," *Harvard Business Review*, 76, 121–131.

David Chen, D., Doumeingts, G., and Vernadat, F. (2008), "Architectures for Enterprise Integration and Interoperability: Past, Present and Future," *Computers in Industry*, 56, 647–659.

de Groote, X. (1994), "The Flexibility of Production Processes: A General Framework," *Management Science*, 40, 933-945.

De Leeuw, A., and Volberda, H. (1996), "On the Concept of Flexibility: A Dual Control Perspective," *International Journal of Management Science*, 24, 121-139.

De Meyer, A., Nakane, J., Miller, J. G., and Ferdows, K. (1989), "Flexibility: The Next Competitive Battle the Manufacturing Future Survey," *Strategic Management Journal*, 10, 135-144.

Denton, D. K. (1994), "The Power of Flexibility," *Business Horizons*, 37, 43-46.

Doumeingts, G., Ducq, Y., Vallespir, B., and Kleinhans, S. (2000), "Production Management and Enterprise Modelling," *Computers in Industry*, 42, 245-263.

Dreiling, A., Rosemann, M., Aalst, W. v. d., Sadiq, W., and Khan, S. (2006), "Model-Driven Process Configuration of Enterprise Systems," *Advanced Information Systems Engineering - CAiSE 2006*.

Du, X. F., Jiao, J. X., and Tseng, M. (2000), "Architecture of Product Family for Mass Customization," in *IEEE International Conference on Management of Innovation and Technology*.

Edwards, G., Deng, G., Schmidt, D. C., Gokhale, A., and Natarajan, B. (2004), *Model-Driven Configuration and Deployment of Component Middleware Publish/Subscribe Services* (3286 ed.),

Evans, J. S. (1991), "Strategic Flexibility for High Technology Manoeuvres: A Conceptual Framework," *Journal of Management Studies*, 28, 69-89.

Eynard, B., Gallet, T., Nowak, P., and Roucoules, L. (2004), "Uml Based Specifications of Pdm Product Structure and Workflow," *Computers in Industry*, 55, 301-316.

Fernandez, W. D., Lehmann, H. P., and Underwood, A. (2002), "Rigour and Relevance in Studies of Is Innovation: A Grounded Theory Methodology Approach," *Proceedings of the Xth European Conference on Information Systems ECIS 2002: Information Systems and the Future of the Digital Economy*.

Fitzgerald, G. (1990), "Achieving Flexible Infon-Nation Systems: The Case for Improved Analysis," *Journal of Information Technology*, 5, 5-11.

Footen, J., and Faust, J. (2008), "Service-Oriented Architecture: Definition, Concepts, and Methodologies," in *The Service-Oriented Media Enterprise*, Boston: Focal Press, pp. 65-146.

Fujita, K. (2002), "Product Variety Optimization under Modular Architecture," *Computer-Aided Design*, 34, 953-965.

Fuxin, F. (2005), "Configurable Product Views Based on Geometry User Requirements," *Computer-Aided Design*, 37, 957-966.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1998), *Design Patterns*, Addison Wesley.

Garavelli, A. C. (2003), "Flexibility Configurations for the Supply Chain Management," *International Journal of Production Economics*, 85, 141-153.

Gebauer, J., and Lee, L. (2008), "Enterprise System Flexibility and Implementation Strategies: Aligning Theory with Evidence from a Case Study," *Information Systems Management*, 25, 71-82.

Gebauer, J., and Schober, F. (2006), "Information System Flexibility and the Cost Efficiency of Business Processes," *Journal of the Association for Information Systems*, 7, 122-122.

Gerwin, D. (1987), "An Agenda for Research on the Flexibility of Manufacturing Processes," *International Journal of Operations & Production Management*, 7, 38-49.

Gerwin, D. (1993), "Manufacturing Flexibility: A Strategic Perspective," *Management Science*, 39, 395-410.

Golden, W., and Powell, P. (2000), "Towards a Definition of Flexibility: In Search of the Holy Grail," *Omega*, 28, 373-384.

Gorod, A., Gandhi, S., Sauser, B., and Boardman, J. (2008), "Flexibility of System of Systems," *Global Journal of Flexible Systems Management*, 9, 21-31.

Goyal, J. (2006), "Challenges to Enterprise Systems Manufacturers: Delivering Flexibility, Managing Complexity, and Providing Optimal Service," http://www.oracle.com/ocom/groups/public/@ocompublic/documents/webcontent/022563.pdf.

Gracanin, D., Singh, H. L., Bohner, S. A., and Hinchey, M. G. (2004), *Model-Driven Architecture for Agent-Based Systems* (3228 ed.),

Gupta, D. (1993), "On Measurement and Valuation of Manufacturing Flexibility," *International Journal of Production Economics Research*, 31, 2947-2958.

Gupta, D., and Buzacott, J. A. (1989a), "Impact of Flexible Machines on Automated Manufacturing Systems," *Annals of Operations Research*, 15, 169-205.

Gupta, Y. P. (1989), "Human Aspects of Flexible Manufacturing Systems," *Production and Inventory Management Journal*, 30-35.

Gupta, Y. P., and Goyal, S. (1989b), "Flexibility of Manufacturing Systems: Concepts and Measurements," *European Journal of Operational Research*, 43, 119-135.

Gupta, Y. P., and Somers, T. M. (1992), "The Measurement of Manufacturing Flexibility," *European Journal of Operational Research*, 60, 166-182.

Hameri, A., and Nihtila, J. (1998), "Product Data Management—Exploratory Study on State-of-the-Art in One-of-a-Kind Industry," *Computers in Industry*, 35, 195-206.

Hammer, M. (2002), "Process Management and the Future of Six Sigma," *IEEE Engineering Management Review*, 34, 56-63.

Hanneghan, M., Merabti, M., and Colquhoun, G. (2000), "A Viewpoint Analysis Reference Model for Concurrent Engineering," *Computers in Industry*, 41, 35-49.

He, W., Ni, Q. F., and Lee, B. H. (2003), "Enterprise Business Information Management System Based on Pdm Framework," in *IEEE International Conference on Systems, Man & Cybernetics*, Washington, D.C., USA.

He, W., Ni, Q. F., Ming, X., and Lu, W. F. (2004), "Product Structure Management for Enterprise Business Processes in Product Lifecycle," in *1th ISPE International Conference on Concurrent Engineering*, Beijing, China.

Heckel, R., and Lohmann, M. (2003), *Model-Based Development of Web Applications Using Graphical Reaction Rules* (2621 ed.),

Heinl, P., et al. (1999), "A Comprehensive Approach to Flexibility in Workflow Management Systems," in *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration*, San Francisco, USA, pp. 79–88.

Henschen, D. (2005), "Business Process Management Is under Construction," *Intelligent enterprise(http://www.intelligententerprise.com/showArticle.jhtml?articleID=165700250&pgno=2)*.

Hill, T., and Chambers, S. (1991), "Flexibility -a Manufacturing Conundrum," *International Journal of Operations & Production Management*, 11, 5-13.

Holland, C. P., and Light, B. (1990), "A Critical Success Factors Model for Erp Implementation," *IEEE Software*, 16, 30-36.

Holschke, O., Rake, J., Offermann, P., and Bub, U. (2010), "Improving Software Flexibility for Business Process Changes," *Business & Information Systems Engineering*, 2, 3-13.

Hu, J., and Grefen, P. (2003), "Conceptual Framework and Architecture for Service Mediating Workflow Management," *Information and Software Technology*, 45, 929–933.

Hyun, J. H., and Ahn, B. H. (1992), "A Unifying Framework for Manufacturing Flexibility," *Manufacturing review*, 5, 251-260.

Jablonski, S., and Bussler, C. (1996), *Workflow Management : Modeling Concepts, Architecture and Implementation*, Sydney: International Thomson Computer Press.

Janitza, D., Lacher, M., Maurer, M., Pulm, U., and Rudolf, H. (2003), "A Product Model for Mass-Customisation Products," *Lecture Notes in Computer Science*, 2774, 1023-1029.

Jensen, T., and Baumgartner, B. (2003), "A Flexible, Multimodality Structured Reporting System Based on Medical and Networking Standards," *International Congress Series*, 1256, 893-899.

Kahn, C. E. (1998), "Self-Documenting Structured Reports Using Open Information Standards," in *Proceedings of the 9th World Congress on Medical Informatics*, Amsterdam, Netherlands, pp. 403-407.

Kaim, W. E., Studer, P., and Muller, P.-A. (2003), *Model Driven Architecture for Agile Web Information System Engineering* (2817 ed.),

Kathuria, R. (1998), "Managing for Flexibility: A Manufacturing Perspective," *Industrial Management & Data Systems*, 98, 246-252.

Keller, G., and Teufel, T. (A. Weinland, trans.) (1998), *Sap R/3 Process-Oriented Implementation*, England, UK: Addison-Wesley Longman.

Kent, S. (2002), *Model Driven Engineering* (2335 ed.),

Kim, C. (1991), "Issues on Manufacturing Flexibility," *Journal of Production Research*, 2, 4-13.

Kim, C., Kim, K., and Choi, I. (1993), "An Object-Oriented Information Modeling Methodology for Manufacturing Information Systems," *Computer Ind. Engng*, 24, 337-353.

King, J. (1997), "Dell Zaps Sap," *Computerworld*, 2.

Koste, L., and Malhotra, M. K. (1999), "A Theoretical Framework for the Dimensions of Manufacturing Flexibility," *Journal of Operations Management*, 18, 75-93.

Langlotz, C. P. (2000a), "Structured Reporting in Radiology," Technical.

Langlotz, C. P. (2000b), "Structured Reporting in Radiology, Society for Health Services Research in Radiology," *News [serial online]*.

Li, H., and Williams, T. J. (1997), "Some Extensions to the Purdue Enterprise Reference Architecture (Pera): I. Explaining the Purdue Architecture and the Purdue Methodology Using the Axioms of Engineering Design," *Computers in Industry*, 34, 247-259.

Li, H., Yang, Y., and Chen, T. Y. (2004), "Resource Constraints Analysis of Workflow Specifications," *Journal of Systems and Software*, 73, 271-285.

Li, M., Wang, J., Wong, Y. S., and Lee, K. S. (2004), "A Collaborative Application Portal for the Mould Industry," *International Journal of Production Economics*, 96, 233-247.

Lindsay, A., Downs, D., and Lunn, K. (2003), "Business Processes--Attempts to Find a Definition," *Information and Software Technology*, 45, 1015-1019.

Lings, B. (2009), "Linking Model-Driven Development and Software Architecture: A Case Study," *IEEE transactions on software engineering*, 35, 83.

Liu, C. Y., Wang, X. K., and He, Y. C. (2004)*Material Process Technology 2004; 139(3):40-43*, 139, 40-43.

Liu, S. (2003), "A Practical Framework for Distributing It Infrastructure," *IT Professional*, 14-20.

Luo, L., and Bai, X. (2005), "Web Services-Based Test Report Generation," *Tsinghua Science &Technology*, 10, 282-287.

Lynch, R. L., and Cross, K. F. (1991), *Measure Up!* , Blackwell, Cambridge, MA.

MacCarthy, B., Brabazon, P. G., and Bramham, J. (2003), "Fundamental Modes of Operation for Mass Customization," *International Journal of Production Economics*, 85, 289-304.

Maksimovic, R., and Lalic, B. (2008), "Flexibility and Complexity of Effective Enterprises," *Strojniski Vestnik-Journal of Mechanical Engineering*, 54, 768-782.

Mandelbaum, M., and Brill, P. H. (1989), "Examples of Measurement of Flexibility and Adaptivity in Manufacturing Systems," *The Journal of the Operational Research Society*, 40, 603-609.

Mannisto, T., Peltonen, H., Martio, A., and Sulonen, R. (1998), "Modelling Generic Product Structures in Step," *Computer-Aided Design*, 30, 1111-1118.

Margaria, T., and Steffen, B. (2009), "Continuous Model-Driven Engineering," *Computer*, 42, 106-109.

Marshall, C., and Rossman, G. B. (1989), *Designing Qualitative Research*, Newbury Park,California: Sage.

Martinho, R. (2010), "Goals and Requirements for Supporting Controlled Flexibility in Software Processes," *Information resources management journal*, 23, 11-26.

McCarty, B., and Cassady-Dorin, L. (1999), *Java Distributed Objects*, USA: Macmillan Computer Publishing.

Melia, K. M. (1996), "Rediscovering Glaser," *Qualitative Health Research*, 6, 368-378.

Nakane, J., and Hall, R. W. (1991), "Holonic Manufacturing: Flexibility - the Competitive Battle in the 1990s. ," *Production Planning and Control*, 2, 2-13.

Narain, R., Yadav, R. Q., Sarkis, J., and Cordeiro, J. J. (2000), "The Strategic Implications of Flexibility in Manufacturing Systems," *International Journal of Agile Management Systems*, 2, 202-213.

Narendra, N. C. (2004), "Flexible Support and Management of Adaptive Workflow Processes," *Information Systems Frontiers*, 6, 247–262.

Nasirin, S., and Birks, D. (2002), "Factors Influencing the Employment of Grounded Theory Approach in Understanding Is Project Implementation Process," *European Conference on Research Methodology for Business and Management Studies*.

Ni, Q., Ming, X., and Lu, W. F. (2003), "Computer-Supported Collaborative Environment for Distributed Product Development," in *International Conference for Agile Manufacturing*, Beijing, Chian.

Ni, Y. (2007), "*The Impact of Information Systems on Business Flexibility from the Managerial Perspective: Multiple Cases of Enterprise Systems Enhancement and Ongoing Changes*," The University of Warwick, Warwick Business School.

Noran, O. (2003), "An Analysis of the Zachman Framework for Enterprise Architecture from the Geram Perspective," *Annual Reviews in Control*, 27, 163-183.

Olexa, R. (2001), "The Father of the Second Industrial Revolution," *Manufacturing Engineering*, 127.

Oliver, A. C., Stampoultzis, G., and Sengupta, A. (2004), "Welcome to Poi - the Apache Software Foundation," *http://jakarta.apache.org/poi*.

Ortiz, A., Lario, F., and Ros, L. (1999), "Enterprise Integration--Business Processes Integrated Management: A Proposal for a Methodology to Develop Enterprise Integration Programs," *Computers in Industry*, 40, 155-171.

Ozer, M. (2002), "The Role of Flexibility in Online Business," *Business Horizons*, 61-69.

Palanisamy, R., and Sushil. (2003), "Achieving Organizational Flexibility and Competitive Advantage through Information Systems," *Journal of Information & Knowledge Management*, 2, 261-277.

Parker, R., and Wirth, A. (1999), "Manufacturing Flexibility: Measures and Relationships," *European Journal of Operational Research*, 118.

PTC (2000), *Windchill Customization Guide for Windchill Release 6.2*, USA:

Qiu, Z. M., and Wong, Y. S. (2007), "Dynamic Workflow Change in Pdm Systems," *Computers in Industry*, 58, 453-463.

Robey, D., and Boudreau, M. C. (1999), "Accounting for the Contradictory Organizational Consequences of Information Technology: Theoretical Directions and Methodological Implications," *Information Systems Research*, 10, 167-185.

Robinson, W. N., and Pawlowski, S. D. (1999), "Managing Requirements Inconsistency with Development Goal Monitors," *IEEE Transactions on Software Engineering*, 25, 816-835.

Rolland, C., and Prakash, N. (2000), "Bridging the Gap between Organisational Needs and Erp Functionality," *Requirements Engineering*, 5, 180-193.

Ross, J. W., and Vitale, M. R. (2000), "The Erp Revolution: Surviving Vs. Thriving," *Information Systems Frontiers*, 2, 233-241.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddi, F., and Lorensen, W. (1991), *Object-Oriented Modeling and Design*, Prentice-Hall.

Sadiq, W., and Orlowska, M. E. (2000), "Analyzing Process Models Using Graph Reduction Techniques," *Information Systems*, 25, 117-134.

Salimifard, K., and Wright, M. (2001), "Petri Net-Based Modelling of Workflow Systems: An Overview," *European Journal of Operational Research*, 134, 664-676.

Sanchez, R., and Mahoney, J. T. (1996), "Modularity, Flexibility, and Knowledge Management in Product and Organisation Design," *Strategic Management Journal*, 17, 63-76.

Sarker, B. R., Krishnamurthy, S., and Kuthethur, S. G. (1994), "A Survey and Critical Review of Flexibility Measures in Manufacturing Systems," *Production Planning and Control*, 5, 512-523.

Sarker, S., and Lee, A. S. (2003), "Using a Case Study to Test the Role of Three Key Social Enablers in Erp Implementation," *Information & Management*, 40, 813-829.

Sarkis, J., and Sundarraj, R. P. (2003), "Managing Large-Scale Global Enterprise Resource Planning Systems: A Case Study at Texas Instruments," *International Journal of Information Management*, 23, 431–442.

Schmenner, R. W., and Tatikonda, M. V. (2005), "Update Manufacturing Process Flexibility Revisited," *International Journal of Operations & Production Management*, 25, 1183-1189.

Schmidt, D. C. (2006), "Model-Driven Engineering," *IEEE Computer*, 39, 25-31.

Schober, F., and Gebauer, J. (2008), "How Much to Spend on Flexibility? Determining the Value of Information System Flexibility."

Scott, J. E., and Vessey, I. (2000), "Implementing Enterprise Resource Planning Systems: The Role of Learning from Failure," *Information Systems Frontiers*, 2, 213-232.

Sethi, A. K., and Sethi, S. P. (1990), "Flexibility in Manufacturing: A Survey," *International Journal of Flexible Manufacturing Systems*, 2, 289-328.

Shen, H., Wall, B., Zaremba, M., Chen, Y., and Browne, J. (2004), "Integration of Business Modelling Methods for Enterprise Information System Analysis and User Requirements Gathering," *Computers in Industry*, 54, 307-323.

Shin, K., and Leem, C. S. (2002), "A Reference System for Internet Based Inter-Enterprise Electronic Commerce," *Journal of Systems and Software*, 60, 195-209.

Shu, Q., and Wang, C. (2005), *Information Modeling for Product Lifecycle Management* (183 ed.),

Silver, M. S. (1991), *Systems That Support Decision Makers: Description and Analysis Chichester*, United Kingdom:Wiley & Sons.

Slack, N. (1987), "The Flexibility of Manufacturing System," *International Journal of Operations & Production Managemen*, 7, 35-45.

Slack, N. (1989), *Focus on Flexibility*, ed. R. Wild, Cassell Educational Ltd.

Smith, H., and Fingar, P. (2003), *Business Process Management (Bpm): The Third Wave*, Meghan-Kiffer Press.

Stevenson, M., and Spring, M. (2009), "Supply Chain Flexibility: An Inter-Firm Empirical Study," *International Journal of Operations & Production Management*, 29, 946-971.

Stohr, E. A., and Zhao, J. L. (2001), "Workflow Automation: Overview and Research Issues," *Information Systems Frontiers*, 3, 281–296.

Sudarsan, R., Fenves, S. J., Sriram, R. D., and Wang, F. (2005), "A Product Information Modeling Framework for Product Lifecycle Management," *Computer-Aided Design*, 37, 1399-1411.

Sun, H. (2000), "Current and Future Patterns of Using Advanced Manufacturing Technologies," *Technovation*, 20, 631-641.

Sun, P., and Jiang, C. (2009), "Analysis of Workflow Dynamic Changes Based on Petri Net," *Information and Software Technology*, 51, 284-292.

Tang, D. B. (2004), "An Agent-Based Collaborative Design System to Facilitate Active Die-Maker Involvement in Stamping Part Design," *International Journal of Production Economics*, 54, 253 - 271.

ter Hofstede, A. H. M., Orlowska, M. E., and Rajapakse, J. (1998), "Verification Problems in Conceptual Workflow Specifications," *Data & Knowledge Engineering*, 24, 239-256.

Thimm, G., Lee, S. G., and Ma, Y.-S. (2006), "Towards Unified Modelling of Product Life-Cycles," *Computers in Industry*, 57, 331-341.

Touzi, J., Benaben, F., Pingaud, H., and Lorré, J. P. (2009), "A Model-Driven Approach for Collaborative Service-Oriented Architecture Design," *International Journal of Production Economics*, 121, 5-20.

Umble, E. J., Haft, R R, and Umble, M. M. (2003a), "Enterprise Resource Planning: Implementation Procedures and Critical Success Factors," *European Journal of Operational Research*, 146, 241–257.

Umble, E. J., Haft, R. R., and Umble, M. M. (2003b), "Enterprise Resource Planning: Implementation Procedures and Critical Success Factors," *European Journal of Operational Research*, 146, 241-257.

Upton, D. M. (1994), "The Management of Manufacturing Flexibility," *California Management Review*.

Upton, D. M. (1997), "Process Range in Manufacturing- an Empirical Study of Flexibility," *Management Science*, 43, 1079-1092.

van der Aalst, W. M. P. (1999), "Formalization and Verification of Event-Driven Process Chains," *Information and Software Technology*, 41, 639-650.

van der Aalst, W. M. P., and Basten, T. (2002), "Inheritance of Workflows: An Approach to Tackling Problems Related to Change," *Theoretical Computer Science*, 270, 125-203.

Vinther, F. (2008), "Extreme Flexibility," *InTech*, 55, 38-41.

Vokurka, R. J., and O'Leary-Kelly, S. W. (2000), "A Review of Empirical Research on Manufacturing Flexibility," *Journal of Operations Management*, 18, 485-501.

Volberda, H. W. (1996), "Toward the Flexible Form: How to Remain Vital in Hypercompetitive Environments," *Organizational Science*, 7, 359-374.

Volberda, H. W. (1999), *Building the Flexible Firm - How to Remain Competitive*, Oxford University Press.

Wada, H., Suzuki, J., and Oba, K. (2008), "A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture," *International Journal of Web Services Research*, 5, 1-31.

Wallace, M., Schimpf, J., Shen, K., and Harvey, W. (2004), "On Benchmarking Constraint Logic Programming Platforms. Response to Fernandez and Hill's 鈥淎 Comparative Study of Eight Constraint Programming Languages over the Boolean and Finite Domains 鈥," *Constraints*, 9, 5-34.

Walsh, E. (2010), "Open Architecture: Versatility through Flexibility," *United States Naval Institute. Proceedings*, 136, 86.

Wang, H., Huang, J. Z., Qu, Y., and Xie, J. (2004), "Web Services: Problems and Future Directions," *Web Semantics: Science, Services and Agents on the World Wide Web*, 1, 309-320.

Whiting, R. (2003), "Money Machines," *Informationweek*, 34-44.

Whittingham, K. (1999), *Openwater—White Paper*, ed. I. R. Division, Zurich Research Laboratory.

Williamson, M. (1997), "From Sap to `Nuts!'," *Computerworld*, 45.

Xiao, L., and Greer, D. (2009), "Adaptive Agent Model: Software Adaptivity Using an Agent-Oriented Model-Driven Architecture," *Information and Software Technology*, 51, 109-137.

Xu, Q., and Jiao, J. (2009a), "Modeling the Design Process of Product Variants with Timed Colored Petri Nets," *Journal of Mechanical Design*, 131, 061009-061009.

Xu, Q., and Jiao, J. R. (2009b), "Design Project Modularization for Product Families," *Journal of Mechanical Design*, 131, 071007-071010.

Yen, H. R., and Sheu, C. (2004), "Aligning Erp Implementation with Competitive Priorities of Manufacturing Firms: An Exploratory Study," *International Journal of Production Economics*, 92, 207-220.

Zelenovich, D. M. (1982), "Flexibility: A Condition for Effective Production Systems," *International Journal of Production Research*, 20, 319-337.

Zhang, Q., and Cao, M. (2002), "Business Process Reengineering for Flexibility and Innovation in Manufacturing," *Industrial Management & Data Systems*, 102, 146-152.

Zhang, Z., Lee, M. K. O., Huang, P., Zhang, L., and Huang, X. (2004), "A Framework of Erp Systems Implementation Success in China: An Empirical Study," *International Journal of Production Economics*.

Zhang, Z., Lee, M. K. O., Huang, P., Zhang, L., and Huang, X. (2005), "A Framework of Erp Systems Implementation Success in China: An Empirical Study," *International Journal of Production Economics*, 98, 56-80.