Chapter 2

Background and Related Work

This chapter provides necessary background information and premises of the thesis, along with related work to it.

2.1 Background and Premises

2.1.1 Enterprise Modeling

One of the generic definition of an *enterprise model* is that "an enterprise model is a computational representation of the structure, processes, information, resources, goals and constraints of a business, government activity or other organizational system. It can be both definitional and descriptive - spanning what should be and what is. The role of an enterprise model is to achieve model-driven enterprise design, analysis and operation" [34].

Since this thesis deals with software system development for enterprises, especially for enterprise back-office applications, it focuses on the process, information and resource aspects of an enterprise model, and the model represents an enterprise as a set of *business processes*.

There are several ways to define a business process such as "a set of logically related tasks performed to achieve a defined outcome" [18] or "a structured, measured set of activities designed to produce a specified output for a particular customer or market" [19].

However, business processes which are dealt with in this thesis are more restricted ones from software system viewpoint, and are defined as " an ordered sequence of activities which transform resources or information". The functionality and behavior of a business process are mainly focused on.

There is another consideration in enterprise modeling which is known as *management layers*. Usually, an enterprise or enterprise organization is composed of

three or four layers, that is, *operational*, *supervising*, *managerial*, and *administrative* layers. (In case of three layers, *supervising* and *managerial* are identical.)

Each layer has its unique perspective of business processes and therefore we need to build different enterprise models for those layers. However, this thesis only focuses on the *operational layer*, since the purpose of enterprise modeling in it is to build the enterprise models for software systems which support business operations in back-office applications.

2.1.2 Software Composition in CBSD

It is called *model based approach* or *model driven development* to implement software systems from requirements models expressed in appropriate ways. Although there are numerous notation methods for enterprise models, such as, Unified Modeling Language (UML) [7], the IDEF notation set [77], Data Flow Diagram (DFD) [20] or Entity-Relationship (ER) diagram [4], the thesis uses Colored Petri Nets (CPN) as the only notation method. CPN are one of the enhancements of ordinary Petri nets, and expressional capability is reinforced by incorporating colors (data types) of tokens and functions on arcs into them.

Component Based Software Development (CBSD) is an approach to promote *componentization* of software systems, that is, making software systems be composed of reusable components. CBSD includes architectures, frameworks, tools and languages which are intended to achieve the above purpose. When those components are available to the general public, they are called Commercial-Off The Shelf (COTS) components or shrinkwrapped software.

Software development in CBSD includes the following four major processes [37].

1. Component qualification
   A process of determining "fitness for use" of components that are being applied in a new system context.

2. Component adaptation
   A process of adapting components for specific requirements, if components are written to meet the different requirements or are based on the different assumption.

3. Assembling components into systems
   A process of integrating components through well-defined infrastructure such as object request broker.

4. System evolution
   A process of updating systems, such as repairing an error or adding new functionality.

This thesis focuses on the above 2 and 3 as the succeeding processes of enterprise modeling.

2.1.3 Adaptability Evaluation of Software Systems

Adaptability of software systems depends on the specified requirements for them, and is defined as the degree of satisfaction of the specified requirements.

Roughly speaking, there are two kinds of requirements [72]. One is the functional type requirements which include functionality and behavior, and the other is non-functional type requirements which include cost, performance, quality, reliability and so forth.

The thesis deals with only the functional type requirements, since enterprise modeling and software composition are performed only from such viewpoints in this thesis.

The evaluation process is composed of software verification tasks. There are two ways for software verification, that is, formal verification and software testing. The thesis only focuses on formal verification using Σ algebra and process algebra. Σ homomorphism is used as a measure of adaptability in Σ algebra, while *bisimulation* is used as a measure in process algebra. They are discussed in Chapter 5.

2.2 Related Work

2.2.1 Enterprise Modeling

There have been many efforts to establish efficient and effective methodologies, frameworks and tools to construct precise enterprise models for both business and software realms. They include CIMOSA [43], PERA [61], [81], ARIS [62] and TOVE [26], [33]. Brief description and characteristics of them are shown below.

(**CIMOSA**)

CIMOSA is the European Open System Architecture of CIM (Computer Integrated Manufacturing), and has been developed as a series of the ESPRIT projects. It provides a consistent architectural framework for both enterprise modeling and enterprise integration.

The CIMOSA modeling framework has three orthogonal principles, and each principle includes several modeling viewpoints in it. The modeling principles and viewpoints defined by CIMOSA are as follows.

1. The derivation principle: which advocate to model enterprises according to the following three modeling viewpoints (or levels).

- requirements definition
  to express business needs as perceived by users.
- design specification
  to build a formal, conceptual executable model of the enterprise system.
- implementation description
  to document implementation details, installed resources, exception handling mechanism, and taking into account system non-determinism.

2. The instanciation principle: which is based on the following three viewpoints.

   - generic
     containing generic building blocks and building block types as the elements of the modeling language to express any model.
   - partial
     containing libraries of partial models classified by industry sectors to be copied and used in particular models.
   - particular
     containing particular model, i.e. company specific models of parts of a generic enterprise.

3. The generation principle: which recommends to model enterprises according to the following basic and complementary viewpoints.

   - function view
     which represents enterprise functionality and behavior (i.e. events, activities and processes) including temporal and exception handling aspects.
   - information view
     which represents enterprise objects and their information elements.
   - resource view
     which represents enterprise means, their capabilities, and management.
   - organization view
     which represents organizational levels, authorities and responsibilities.

(PERA)

The Purdue Enterprise Reference Architecture (PERA) has been developed at the University of Purdue. It provides a complete methodology for enterprise modeling in manufacturing industry. PERA is characterized by its layering structure, which is composed of:

1. Concept layer (mission, views and values)
   presents or proposes production entity including product and operational policies.

2. Definition layer (functional requirements)
   defines physical production manufacturing functional models and manufacturing functional network.

3. Specification layer (functional design)
   specifies manufacturing architecture, information architecture and organization architecture.

4. Detailed design layer
   makes detailed physical design of "manufacturing equipment architecture", "human and organizational architecture" and "information system architecture".

5. Manifestation layer
   describes "construction", "plant testing", "commissioning", "staffing", "training", "procurement" and "equipment installation".

6. Operation layer
   includes "continuing development", "maintenance", "continuing organizational development", "continuing skill and human relations training" and "maintenance".

The methodology starts first with the identification of the enterprise entity. PERA is supported by simple graphical formalism and easy-to-understand textual manuals.

(ARIS)

Architecture for Integrated information System (ARIS) has been developed at the University of Saarbrüken. While the previous two methodologies mainly deal with manufacturing enterprises, this methodology deals with more generic business-oriented issues of enterprises. ARIS models enterprises from the following four views.

1. The function view
   which determines the functional structure, process requirements and processing forms of an enterprise.

2. The organization view
   which determines the structure of an enterprise expressed by organization, network topology and physical network implementation.

3. The data view
   which determines the data semantics for functions, in terms of extended ER model.

4. The control view
   which determines the bridges between functions, organization and data.

Each view has three description levels, that is, *requirements definition* level, *design specification* level and *implementation description* level. ARIS defines the modeling procedure step-by-step, and Computer Aided Software Engineering (CASE) tools are available.

(TOVE)

TOronto Virtual Enterprise (TOVE) modeling project has been carried by University of Toronto. The goal of it is to create *enterprise ontologies*. An *ontology* in this project is defined as "a formal description of entities and their properties which forms a shared terminology for the objects of interest in the domain, along with definition for the meaning of each of the terms".

TOVE is a knowledge based approach to enterprise modeling, and it incorporates the techniques from Artificial Intelligence (AI) realm, such as Prolog, Knowledge Interchange Format (KIF), completeness theorem and so forth.

The TOVE ontologies currently cover "activity", "time and causality", "resources", "cost", "quality" and "organization structure" which are dealt with enterprise modeling. TOVE defines the process for developing an ontology as follows:

1. Motivating scenarios
   The development of ontologies is motivated by scenarios that arise in the applications. Those scenarios often have the form of story problems, and are provided with a set of intuitively possible solutions.

2. Informal competency questions
   A set of queries will arise which place demands on an underlying ontology. Those queries can be requirements that are in the form of questions that an ontology must be able to answer.

3. Specification in first-order logic - terminology
   Once informal competency questions have been posted for the proposed ontology, the terminology of the ontology is specified using first-order logic.

4. Specification in first-order logic - axioms
   The axioms in the ontology specify the definition of terms in the ontology, and constraints on their implementation.

5. Completeness theorem
   This step defines the conditions under which the solutions to the questions are complete. This forms the basis for completeness theorem for the ontology.

(The differentiation from the above related work)

While the above four methodologies provide us with systematic ways to build enterprise models from some "given" or "known" model units or elements which are supposed to compose an enterprise, this thesis provides a way to identify those units from various pieces of knowledge owned by many domain-experts. In addition, the above methodologies aim to be used in traditional waterfall software development, therefore they are not always adequate in recent CBSD environments. On the other hand, the proposed approach in this thesis devotes itself to enterprise modeling for CBSD environments.

2.2.2 Software Composition in CBSD and Adaptability Evaluation

Since software composition and adaptability evaluation of the implemented software systems are often discussed together in one methodology or research, related work on those topics is summarized in the single section together.

Two related researches to this thesis are introduced. One is the research on "COTS-Based System" (CBS) at Software Engineering Institute (SEI) in Carnegie Mellon University (CMU) [17], [37], and the other is CHAIMS (Compiling High-level Access Interface for Multi-site Systems) project at Stanford University [6].

(CBS)

Intensive and vast studies on CBSD have been done at SEI in CMU, as the CBS project. Those studies include "*product and technology evolution*", "*COTS usage risk evaluation*", "*legacy system evolution and migration*", "*architecture and integration*", "*software process*" and "*design and software engineering*".

Many practical methods and guidelines have been provided with experiences in real world applications. Their software composition framework (referred to as a CBS framework) is composed of four stages, that is:

1. Qualification to discover interface and fitness for use of COTS.

Typically, products are described in terms of interfaces that provide access to functionality. Here, standards may provide a frame of reference for comparing the product to generally accepted capabilities. Various approaches have been developed for evaluating products in terms of their interfaces. However, to determine the fitness of a product for a given use, consideration must be given to more than just the interfaces the product provides. Aspects of performance, reliability, flexibility, etc., as well as the implicit assumptions made by the product about the operating environment must be considered.

2. Adaptation to remove architectural mismatch.

   Once the relevant properties of a component have been discovered, it is possible to identify which properties exhibited by a component are in conflict with other components, or with a system design. These conflicts, or mismatches, must be repaired through component adaptation. Once the mismatches between components have been removed, it is possible to assemble them into systems, and to evolve the system through re-assembly with different components. The reference model assigns a prominent role to software architecture in supporting both assembly and re-assembly.

3. Composition into a selected architectural system.

   The engineering of COTS-based systems continues to involve significant technical risk. A good indicator of the as-yet unresolved difficulties involved in building COTS-based systems is the *glue code* used to integrate components. This code is often ad hoc and brittle, but it is needed to repair mismatched assumptions that are exhibited by the components being integrated.

4. Evolution to update components

   Organizations implementing CBS strategies for new or legacy systems must consider not only immediate system requirements but also the unceasing evolution of computing and software technology.

Few formalism is incorporated into their approach, and so specific composition mechanism is.

(The differentiation from CBS)

Adaptability evaluation of software systems is performed in the above step 2 at each component level, and no system-wide adaptability is considered. On the other hand, this thesis aims fully formalized approach in software composition, along with component selection. The adaptability evaluation of software systems is performed at both each component level and system-wide level.

However, the thesis only deals with functionality and behavior of software systems, and do not refer to such topics as software process, component qualification or legacy system evolution.

(CHAIMS)

CHAIMS aims at providing total development environment based on the concept *megaprogramming* [80], which includes "compilers", "wrappers", "inter-module protocols" called CPAM (CHAIMS Protocol for Autonomous Megamodules), and "development workbenches". It supports recent inter-module communications in distributed systems, such as, CORBA, DCE, DCOM, RMI, and so forth.

Megaprogramming which is the base concept of CHAIMS is defined as "a technology that interconnects large, independent and isolated information management systems from different organizations so that they function together as one unified system". The granularity of each components or building block could be much coarser than software components.

The architecture of CHAIMS can be loosely divided into the following four elements.

1. Service Modules

   The service modules available for use exist independently of CHAIMS. Some are in existence already, motivated by the need to support legacy code. Others will be developed, some with knowledge of CHAIMS (and thus support native CHAIMS primitives) and some with no knowledge of CHAIMS. Wrapper services will provide translation between CHAIMS primitives as specified by the megaprogrammer and services which do not support native CHAIMS primitives. We refer to a service as a megamodule, as we expect these services to be large, typically remote, and operating autonomously.

2. Wrapper

   A wrapper facilitates the translation of messages in the CHAIMS language to the interface the megamodule provides. Some megamodules may support only synchronous invocation. In such cases the wrapper also provides the infrastructure such that the wrapped megamodule appears to support asynchronous invocation to the outside world. A wrapper need not reside at the same site as the megamodule. Furthermore, the distribution layer protocol between the CHAIMS client and the wrapper, and the wrapper and the megamodule will often be different. A single wrapper can be used for more than one megamodule and in such cases it will provide a variety of useful mediation services in addition to the functionality already mentioned above.

3. Distribution Layer

   The distribution layer, which is used to convey the sequence of CHAIMS primitive messages generated by the CHAIMS compiler to the desired megamodules, can be one of many possible distribution protocols developed for client-server systems. Examples include CORBA, DCE, DCOM, JavaRMI, etc. Any distribution protocol can be used, as long as there is a runtime library which allows the CHAIMS compiler to create the appropriate interface to the distribution layer. The distribution layer is of no concern to the megaprogrammer but is the element of the architecture most critical to distributed computing and thus forms a major component of the CHAIMS architecture.

4. Application Megaprogram Written in CHAIMS

   The CHAIMS compiler takes the program specified by the megaprogrammer and generates an appropriate sequence of CHAIMS primitive messages compliant with the distribution layer being used. The compiler also optimizes the sequence. Note that the various megamodules may be using different distribution layer protocols so the messages generated by the CHAIMS compiler may be in different protocols depending upon which megamodule is involved.

(The differentiation from CHAIMS)

While CHAIMS intends to provide infrastructure and mechanism, such as compiler, wrapper or protocols, to construct highly componentized software systems, this thesis aims to provide formalized analysis and design methodology to construct software systems from enterprise models.