An Approach to Architecture-Centric Domain-Specific Modelling and Implementation for Software Development and Reuse

PhD Thesis

Qing Duan

Software Technology Research Laboratory

De Montfort University

2010

To *my husband*, Le, Jin *and*

my mum, Wang, Zhu *and my dad*, Duan, Youde

for their loves and supports

Declaration

I declare that the work described in this thesis was originally carried out by me during the period of registration for the degree of Doctor of Philosophy at De Montfort University, U.K., from July 2004 to April 2010. It is submitted for the degree of Doctor of Philosophy at De Montfort University. Apart from the degree that this thesis is currently applying for, no other academic degree or award was applied for by me based on this work.

Acknowledgements

My deepest gratitude goes to my supervisor, Professor Hongji Yang, for his guidance, support and encouragement throughout my PhD career. He always provided me with many sound comments and suggestions for the improvement of the thesis. I am grateful for his leading role fostering my academic, professional and personal growth.

I would like to thank colleagues in Software Technology Research Laboratory at De Montfort University, for their support and feedback, and for providing such a stimulating working atmosphere, Professor Hussein Zedan, Dr. Shaoyun Li, Dr. Amelia Platt, Dr. Francois Siewe and many other my colleagues. Especially, I would like to express my deep appreciation to Dr. Feng Chen, for his invaluable advice and constant support.

During my work at the Information Technology Institute of Yunnan University, I had the opportunity to design software architectures, take the development of business application integration. The work eventually resulted in this thesis. I would like to thank Prof. Hua Zhou, Prof. Hongzhi Liao and my group members, Prof. Zhihong Liang, Junhui Liu and Xingping Sun, and all my colleagues.

I also want to thank the Graduate School Office at De Montfort University for their outstanding management and warm help during my study.

I thank my parents-in-law, thank my brother Duan, Feiyao and his wife, and thank all my family members, for their memorable support and encouragement that are too precious to forget. I am especially thankful to my parents and my husband, for their support during my PhD study. Without their support, the effort would have been unbearable. Their love empowers me to steer through ups and downs and helped me reach this point today. This thesis is dedicated to them.

Abstract

Model-driven development has been considered to be the hope of improving software productivity significantly. However, it has not been achieved even after many years of research and application. Models are only and still used at the analysis and design stage, furthermore, models gradually deviate from system implementation.

The thesis integrates domain-specific modelling and web service techniques with model-driven development and proposes a unified approach, SODSMI (Service Oriented executable Domain-Specific Modelling and Implementation), to build the executable domain-specific model and to achieve the target of model-driven development. The approach is organised by domain space at architectural level which is the elementary unit of the domain-specific modelling and implementation framework. The research of SODSMI is made up of three main parts:

Firstly, xDSM (eXecutable Domain-Specific Model) is proposed as the core construction for domain-specific modelling. Behaviour scenario is adopted to build the meta-modelling framework for xDSM.

Secondly, XDML language (eXecutable Domain-specific Meta-modelling Language) is designed to describe the xDSM meta-model and its application model.

Thirdly, DSMEI (Domain-Specific Model Execution Infrastructure) is designed as the execution environment for xDSM. Web services are adopted as the implementation entities mapping to core functions of xDSM so as to achieve the service-oriented domain-specific application.

The thesis embodies the core value of model and provides a feasible approach to achieve real model-driven development from modelling to system implementation which makes domain-specific software development and reuse coming true.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AGOS	Atomic Group of dOmain-specific web Services
AGOSOF	suppOrt Framework of AGOS
AS&MC	Action Specifications and Model Constraints
ASL	Action Specification Language
BLEF	Behaviour Logic Execution Framework
BLEU	Behaviour Logic Execution Unit
BNF	Backus-Naur Form
BS	Behaviour Scenario
CWM	Common Warehouse Meta-model
DSM	Domain-Specific Modelling
DSMEI	Domain-Specific Model Execution Infrastructure
DSML	Domain-Specific Modelling Language
DSPROF	PROvider Framework of Domain application web Services
DSL	Domain-Specific Language
EBNF	Extended Backus-Naur Form
ECU	Execution Control Unit
GME	Generic Modelling Environment
MDA	Model Driven Architecture

MDD	Model Driven Development
MMLs	Modelling Maturity Levels
MOF	Meta Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
SODSMI	Service Oriented executable Domain-Specific Modelling and Implementation
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Service Description Language
XDML	eXecutable Domain-specific Meta-modelling Language
xDSM	eXecutable Domain-Specific Model
XMI	XML metadata Interchange
XML	eXtensible Markup Language
xUML	eXecutable Unified Modelling Language
xtUML	eXecutable and Translatable Unified Modelling Language

Chapter 1

Introduction

1.1 Motivation and Problem Description

Software is the spirit of a computer system. It has substantial impacts on success in business today. However, faced with increasing demands and more challenging market pressures, software systems become more and more large and complex. The traditional software development technologies are insufficient for ensuring a successful outcome that fulfills requirements and quality goals set out [39]. The complexity, variety and changeability make the large software projects have staggering failure rates: difficult to maintain, low dependability, high cost and the longer time-to-market. The Standish Report [51] states that nearly a third of projects are cancelled before completion and more than half suffer from serious cost overruns.

Efficiency and quality software development is a matter of the utmost concern of the computer society. During the sixty years from the first computer coming in 1946, the programming language goes through from machine language, assembly language, to advanced language, the third-generation programming language. In the era of advanced language, the development method goes from Structured Development to Object-Oriented Development, then to MDD (Model Driven Development). Each evolution of software development improves the development efficiency, upgrades software quality and maintainability. At the same time, it makes the developer face the problem domain more intuitively, shields the complexity of the development, and enhances the flexibility and retractility of the system.

Software development is switched from code-centric to model-centric with MDD. The model is not only an analysis and design specification, but also a software product which can be automatically transformed into the executable system. MDA (Model Driven Architecture) presented by OMG (Object Management Group) in 2002 is the most representative MDD standardisation system. In OMG blueprint, a series of standards of UML、MOF、XMI、CWM and so on separately resolve the problems of MDA model construction, model extension, model exchange and model transformation. OMG group attempts to expand the scope of application of MDA through the standardisation definition. At the same time, IT vendors can feel free to construct their in-house modelling language along with the mapping from the model to the executable code, so as to ultimately realise the transformation from model to the final executable system.

In 1986，Frederick Brooks proposed “The Silver Bullet Law [12]” and predicted that “Within a decade, there is no single software engineering progress that can improve software productivity by an order of magnitude[12]”. However, even to these days, the industry has not broken through the conclusion. Represented by MDA, MDD has been considered to be the hope of dissolving the silver bullet. After many years of research and application, MDD has not been achieved either. The model is still just used as an aided design tool for software development at the analysis and design stage. Even more seriously, with in-depth software development, the code implementation gradually becomes dominant. The model and the code implementation essentially need to be synchronism updated for maintaining consistency by the designer. But in many cases, the abstractability of the model and the role of the aided design tool make the system model not be updated in time, especially in the software maintenance period. Models gradually deviate from system implementation, which observably reduces the effect of models and makes MDD fall through. Even the agile software development method comes forth in recent years and put emphasis more particularly on prototype practice as well as ignores the documentation and modelling

[7].

A central tenet of modern computer technology adoption has been the promise of reuse, but this has proved difficult to deliver in practice. The reuse mechanisms and fine-grained abstractions offered by object-orientation are rarely sufficient for the development of large software systems. There is a necessary trade-off between reusability and tailorability [39] because users' requirements cannot be effectively anticipated.

Software architecture is a discipline that is able to connect and integrate the various stakeholders, activities, and products involved in software engineering. Software architecture also allows engineers much greater control over and deeper insight into their systems earlier in the development process and can foster early identification and avoidance of problems. As a result, software architecture can help steer the project toward success rather than stumble into failure due to a lack of understanding [39].

Software architecture describes the high-level structure of a software system, and can be used for design, analysis, and software evolution tasks. However, existing tools decouple architecture from implementation, allowing inconsistencies to accumulate as a software system evolves. Because of the potential for inconsistency, engineers evolving a program cannot completely trust the architecture to describe the properties or structure of the implementation accurately.

In response to those challenges, in order to cut the time and cost of development and maintenance, reduce the complexity and invisibility, the methodology with higher abstraction (architecture-centric and model driven approach) for software development has to be pursued. The major topics of this thesis are described as follows:

- To propose an approach to architecture-centric domain-specific modelling and implementation for domain-specific software development and reuse;

- To construct the executable domain-specific model -- xDSM;
- To design the modelling language – XDML for describing xDSM models;
- To design and instantiate the domain-specific model execution infrastructure -- DSMEI;
- To achieve the domain-specific modelling process and the implementation framework;
- To build web service oriented applications with domain-specific implementation framework.

1.2 Original Contributions

In the thesis, a unified approach, SODSMI (Service Oriented executable Domain-Specific Modelling and Implementation) is proposed in the context of MDD, which integrates domain-specific modelling and web service techniques for achieving domain-specific software development and reuse. The original contributions of this thesis are as follows:

C1: The thesis presents the framework of service oriented executable domain-specific modelling and implementation: the domain-specific modelling method is employed to build xDSM (eXecutable Domain-Specific Model); web services are adopted as the core functional implementation entities of xDSM executed on the support of DSMEI (Domain-Specific Model Execution Infrastructure). xDSM can be transformed into the service-oriented domain-specific application by parsing and executing the behaviour logic of xDSM in DSMEI.

C2: Guided by MML5 (Modelling Maturity Levels 5) standard, XDML language (eXecutable Domain-specific Meta-modelling Language) is defined to describe xDSM meta-model and xDSM application model. XDML language integrates the well-defined behaviour semantics to support domain-specific behaviour modelling. The concrete syntax of action specifications and model constraints is constructed on the basis of behaviour semantics, which is used to define the dynamic behaviours of models.

C3: BS (Behaviour Scenario) is proposed as the core of behaviour modelling to describe system behaviours according to system objectives by decoupling behaviour logic and computational logic. BS is constructed from the view of the domain behaviour process. It is represented as the diagram of behaviour logic. The control flow and data flow of behaviour are defined and restricted by AS&MC (Action Specifications and Model Constraints) syntax.

C4: The extension mechanism of xDSM meta-model, which is a round trip from meta-models to application models, is proposed. Based on the primary meta-model of BS, the extension mechanism of xDSM meta-model is realised by the way of using application modelling for the meta-model and the method of meta-level promotion, and the xDSM meta-modelling framework is proposed to extend and construct xDSM meta-model by the way to assemble.

C5: DSMEI is designed and instantiated as the execution environment of xDSM. It utilises BLEF (Behaviour Logic Execution Framework) to interpret and execute the complied xDSM application model, and provides end users with the xDSM model execution application interface by the way of web services to achieve model-driven development.

C6: Web services model based on business document exchange is proposed to design and realise DSPROF (PROvider Framework of Domain application web Services) and AGOSOF (suppOrt Framework of AGOS) for xDSM model execution. On one hand, the dynamic publishing and calling of domain application web services are realised; on the other hand, the virtualisation of AGOS services is realised.

C7: Domain space is proposed as the elementary unit of the domain-specific modelling process and implementation framework. The reuse and composition of domain spaces are realised by the flexible architecture of domain space on the framework of service oriented executable domain-specific modelling and implementation. It makes software reuse at the domain level, realises the reuse of domain knowledge, and openly extends the range and scale of domain-specific model and its implementation.

1.3 Research Methods

The thesis concentrates on the approach of architecture-centric domain-specific modelling and implementation. This section describes the research methods applied in this thesis, which links the knowledge coming from research to the practical outcomes. The research field in this thesis belongs to software engineering aiming to the successful production of domain-specific software and its reuse. The research method applied in this thesis is the combination of empirical and constructive research that is of both high practical utility and academic merit. The basic methods used in this thesis are summarised as follows:

1) Methodology: a methodology is proposed in the thesis for architecture-centric domain-specific modelling and implementation for domain-specific software development and reuse, which links models and system implementation.

2) Observation and analysis: the thesis integrates domain-specific modelling and web service techniques with model-driven development and proposes a unified approach, SODSMI (Service Oriented executable Domain-Specific Modelling and Implementation), to build the executable domain-specific model and achieve the target of model-driven development.

3) Investigation: the thesis studies domain-specific modelling and DSML (Domain-Specific Modelling Language), analyses similarities and differences between domain-specific modelling and the universal modelling such as UML, and investigates the feasibility of the executability of domain-specific models.

4) Modelling: the thesis studies MMLs (Modelling Maturity Levels) with the current software models. Furthermore, the thesis designs the modelling language to construct the executable model according to MMLs 5 through behaviour modelling.

5) Execution infrastructure design: a model itself can not be independently executed. It must be enforced in a certain execution environment. The thesis designs the model execution infrastructure by the modelling language to support the model execution directly.

6) Implementation support: through decoupling behaviour logic and computational logic, system implementation details are encapsulated into web services. Therefore, the thesis proposes the dynamic calling and providing mechanisms based on web service architecture.

7) Extension: domain space is proposed to organise domain-specific modelling and implementation. Domain space is the elementary unit of our approach, which can be reused and assembled so as to support the reuse and composition of domain knowledge at architectural level.

1.4 Success Criteria

The main criterion for the success of the approach to architecture-centric domain-specific modelling and implementation is how to realise model driven software development to create application software. The following criteria are given to judge the success of the research described in the thesis:

- How to make model driven software development from the perspective of domain-specific?
- What is an executable domain-specific model?
- How to describe an executable domain-specific model?
- How to make a domain-specific model executed?
- How to transform application models into the service-oriented domain-specific applications?
- How to realise model-driven software reuse?

1.5 Thesis Outline

The thesis is organised as follows:

- ➢ Chapter 1 describes the motivation and problem, and gives the original contribution, research methods and success criteria of the thesis.
- ➢ Chapter 2 introduces the background of the thesis including software engineering, software architecture, software reuse, models and traditional domain engineering. In the section of models, the executability of different maturity models is investigated and MMLs 5 standard is involved as the guidance throughout the work.
- ➢ Chapter 3 introduces and discusses the background and the state of the art of the related fields including MDD, MDA system and the executability of MDA, DSM, web services and web service composition techniques, etc.

- Chapter 4 raises the problems emerging from model to system implementation and discusses why use the DSM (Domain-Specific Modelling) method as the roadmap. The core idea of the thesis, the framework of SODSMI is describe in this chapter.

- Chapter 5 designs xDSM (eXecutable Domain-Specific Model), and adopts BS (Behaviour Scenario) as the core of xDSM to build the meta-modelling framework for xDSM.

- Chapter 6 defines XDML (eXecutable Domain-specific Meta-modelling Language), which is used to describe xDSM meta-model and xDSM application model, and make xDSM application model executed by DSMEI ultimately. In this chapter, the abstract syntax, the concrete syntax and AS&MC concrete syntax of XDML are described in detail.

- Chapter 7 designs and instances DSMEI (Domain-Specific Model Execution Infrastructure). It gives web service model based on the exchange of business documents, which is used as the basis for designing and implementing DSPROF (PROvider Framework of Domain application web Services) and AGOSOF (Support framework of Atomic Group of dOmain-specific web Services) of DSMEI.

- Chapter 8 proposes domain space which is the elementary unit of the domain-specific modelling and implementation framework. Domain-specific modelling process and implementation framework are introduced for guiding the construction and execution of the domain-specific model.

- Chapter 9 gives two case studies of online shopping system and conference registration system based on mobile. The chapter focuses on xDSM modelling and illustrating the domain-specific modelling process and the implementation framework with these application cases.

- Chapter 10 draws the conclusions, revisits the success criteria of this thesis and discusses the future work.

Chapter 2

Background

2.1 Software Engineering

The IEEE has developed a comprehensive definition of software engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

As Roger S. Pressman states in [104] , software engineering is a layered technology, which has three elements: (1) methods, which provide the techniques for building software including the design of data structures, program architecture, and algorithmic procedure, coding, testing, and maintenance; (2) tools, which provide automated or semi-automated support for methods; and (3) processes, the glue that holds the methods and tools together and enables rational and timely development of computer software (i.e., they define the sequence in which methods would be applied, the deliverables, the controls that help assure quality and coordinate change, and the milestones that enable software managers to assess progress).

The foundation for software engineering is the software process. SEI (Software Engineering Institute) has defined five levels to characterise the maturity of a software development organisation as CMM (Capability Maturity Model):

1. Initial -- ad hoc activities; dependence on the heroic efforts and skills of key individuals.
2. Repeatable -- each project has a well-defined software life cycle, but different models are used for different projects; success is predictable for similar projects.
3. Defined -- uses a documented model for all activities; model is customised at the beginning of each project.
4. Managed -- metrics are defined for activities and deliverables; data is collected during the project to quantify progress
5. Optimised -- measurement data are used to improve the model.

There have been many models for software engineering. The choice of the right model is based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required. Since software development is large and complex work, a phased approach to control it is necessary. The software life cycle is a general model of the software development process, including all the activities and work products required to develop a software system. A software life cycle model is a particular abstraction representing a software life cycle. In this work, a variety of life cycle models are surveyed, most of which focus exclusively on the development processes.

2.1.1 Generic View of Software Engineering

A generic view of software engineering can be obtained by examining the process of software development [104]. The process contains three generic phases, regardless of the software engineering model chosen: the definition, development, and maintenance phases that are encountered in all software development.

The definition phase focuses on what (i.e., the software developer attempts to identify what information is to be processed, what function and performance are desired, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system). The production of this phase is software architecture, which will keep up impacting the following phases.

The development phase focuses on how (i.e., the software developer attempts to describe how the software architecture and associated data structures are to be designed, how procedural details are to be implemented, how the design will be translated into a programming language, and how testing will be performed).

The maintenance phase focuses on change that is associated with error correction, adaptations required as software environment evolves, and modifications due to enhancements brought about by changing customer requirements [129]. The maintenance phase reapplies the steps of the definition and development phases but does so in the context of existing software. The large cost associated with software maintenance is the result of the fact that software has proved difficult to maintain. Early systems tended to be unstructured and ad hoc. This makes it hard to understand their underlying logic. System documentation is often incomplete, or out-of-date. With current methods it is often difficult to retest or verify a system after a change has been made. Successful software will inevitably evolve, but the process of evolution will lead to degraded structure and increasing complexity [63, 9, 81].

2.1.2 Evolutionary Software Process Models

1. Prototyping

As a software product is being developed, the view of developers is divergent from the view of clients. Developers focus on design and implementation while clients focus on requirements. The prototyping model enables the developer to create a prototype of the software to be built to allow problems and requirements to be seen quickly [14]. Prototyping begins with requirements gathering, where developers and customers meet and define the overall objects for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. A quick design then occurs. The quick design focuses on a representation of those aspects of the software visible to the user. The quick design leads to the construction of a prototype. The prototype is evaluated by the customer or user and is used to refine requirements for the software to be developed. A process of iteration occurs as the prototype is "tuned" to satisfy the need of the customer, while at the same time enabling the developer to understand better what needs to be done.

2. Spiral Model

Barry Boehm et al. devised the spiral model to address the weaknesses of the waterfall model [10], especially its lack of resilience in the face of change. The spiral model focuses on addressing risks incrementally by repeating the waterfall model in a series of cycles or rounds.

The spiral model is an improvement on the waterfall model, as it provides for multiple builds and provides several opportunities for customer involvement. However, it is elaborate, difficult to manage, and does not keep all workers occupied during all phases.

3. Iterative and Incremental Development – UML Based Software Life Cycle

UML (Unified Modelling Language) based software development is a famous example of an iterative and incremental software development process. Its designers, Ivar Jacobson et al. characterise the process [49] as:

Use-case driven -- The use case model describes the complete functionality of the system. Use cases are used as a primary product for establishing the desired behaviour of the system, for verifying and validating the system architecture, for testing, and for communicating among the stakeholders of the project.

Architecture-centric -- The software architecture represents the most significant static and dynamic aspects of the system -- the platform on which the software is to run, reusable components and frameworks available, deployment considerations, legacy systems, and non-functional requirements. A system architecture is used as a primary product for conceptualising, constructing, managing, and evolving the system under development.

Iterative and incremental -- The software development project is divided into mini-projects, each of which is an iteration that results in an increment. Each iteration deals with the most important risks and realises a group of use cases that together extend the usability of the product as developed so far. The iterative makes one that involves managing a stream of executable releases, and the incremental makes one that involves the continuous integration of the system architecture to produce these releases.

4. Component-Based Development Model

Over recent years there has been a move towards component-based architectures and software development, reuse and the use of COTS [47]. One of the drivers of this trend is an expectation that increased use of such components will increase system development productivity and response time together with system quality, reliability and evolvability.

Component-based development model absorbs many of the characteristics of Spiral Model. It is a reuse-supporting approach to construct application systems from the pre-packaged software components called classes. This model is more suitable for object-oriented software development, but difficult to be adopted by the classical (structured) software development methodology. The component-based development model leads to software reuse and provides software engineers with a number or measurable benefits.

5. The Fourth Generation Technique

The fourth-generation technique (4GT) model encompasses a broad array of software tools that have one thing in common: each enables the software developer to specify some characteristic of the software at a high level [30]. The tool then automatically generates source code based on specifications written by developers. The 4GT paradigm for software engineering focuses on the ability to specify software to a machine at a level that is close to natural language or in a notation that imparts significant function, but it tends to be used in a single, well-defined application domain. Also the 4GT approach reuses certain elements, such as existing packages and databases rather than reinventing them.

6. MDD (Model Driven Development)

MDD (Model Driven Development) [70] is a new software engineering method which is developed following the object-oriented development methods. It focuses on system modelling based on the best practices to construct software system models. Models are used to guide requirements analysis, system design, code design, system test, and system maintenance at various phases of software development. MDD is the core idea and the target of the thesis. It will be introduced in detail in 3.1.

2.2 Software Architecture

Since the late 1980’s software architecture has been recognised as an important independent area of research for developing and reusing software. Software architecture addresses techniques and approaches for easing difficulties associated with development of large-scale software systems [112, 38, 23, 100].

2.2.1 Software Architecture

While software architecture has long been recognised that finding an appropriate architectural design for a system is a key element of its long-term success, current practice for describing architectures is typically informal and idiosyncratic. Architectural structures are often described in terms of idiomatic patterns that have emerged informally over time. For example, typical descriptions of software architectures include statements such as these:

Definition 1. Garlan & Shaw Model [105]:

SA = {components, connectors, constraints}.

Components can be a group of code, for example, a procedure module, or an independent program such as SQL server for a database. Connectors represent interactions between components, for example, procedure call, pipes and RPC. An overall architecture also includes some constraints.

Definition 2. Bass & Clements & Kazman Model [4]:

SA = {elements, externally visible properties, relationships}.

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. “Externally visible” properties refer to those assumptions other elements can make about an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

Software architecture in the thesis is therefore summarised as follows:

1. Software architecture deals with the design and implementation of the high-level structure of the entire software system. The software architecture of a system is an product. It is the result of the software design activity.

2. A software architecture is a description of elements of a software system and the relationships between them. Elements and relationships are recognised as the fundamental ingredients of software architecture.

Software architecture is concerned with a higher-level abstraction and related to more complex systems. Software architectures consist of elements and relationships to describe the structure and topology of a software system.

2.2.2 Architectural Styles and Patterns

Software architecture may be explored at different levels of abstraction. Shaw and Garlan explored various structural models called architecture styles, which were commonly used in software and then examined quality attributes related to each style [112]. At a lower level of abstraction than style, Shaw and Garlan identified architectural patterns that commonly occur in various design problem domains such as client-server architectures, proxies, etc. In theory, these architecture patterns can be defined by applying a combination of architecture styles.

Using architecture patterns, reference architectures for an application domain or a product line can be built. These architectures embody application domain-specific semantics and quality attributes inherited from the architecture patterns. Application architectures may be created using domain architectures. Examples of domain architectures are reported in [38].

Platform architectures are middleware on/with which applications and components for implementation of an application can be developed. Examples of these are CORBA, COM+, and J2EE. A platform architecture selected for implementation of applications in a domain may influence the architectural decisions for a domain architecture. For example, transaction management is supported by most of platform architectures and a domain architecture may use facilities provided by the platform architecture selected for the domain.

2.2.3 Business Goals, System Objectives and Architecture

From a business perspective the following goals can be defined for products, having impact on the software architecture within such a product [60, 50]:

- Short time-to-market;
- Low cost of product;
- High productivity of organisation;
- Adequate predictability of process;
- High reliability of product;
- High quality of products.

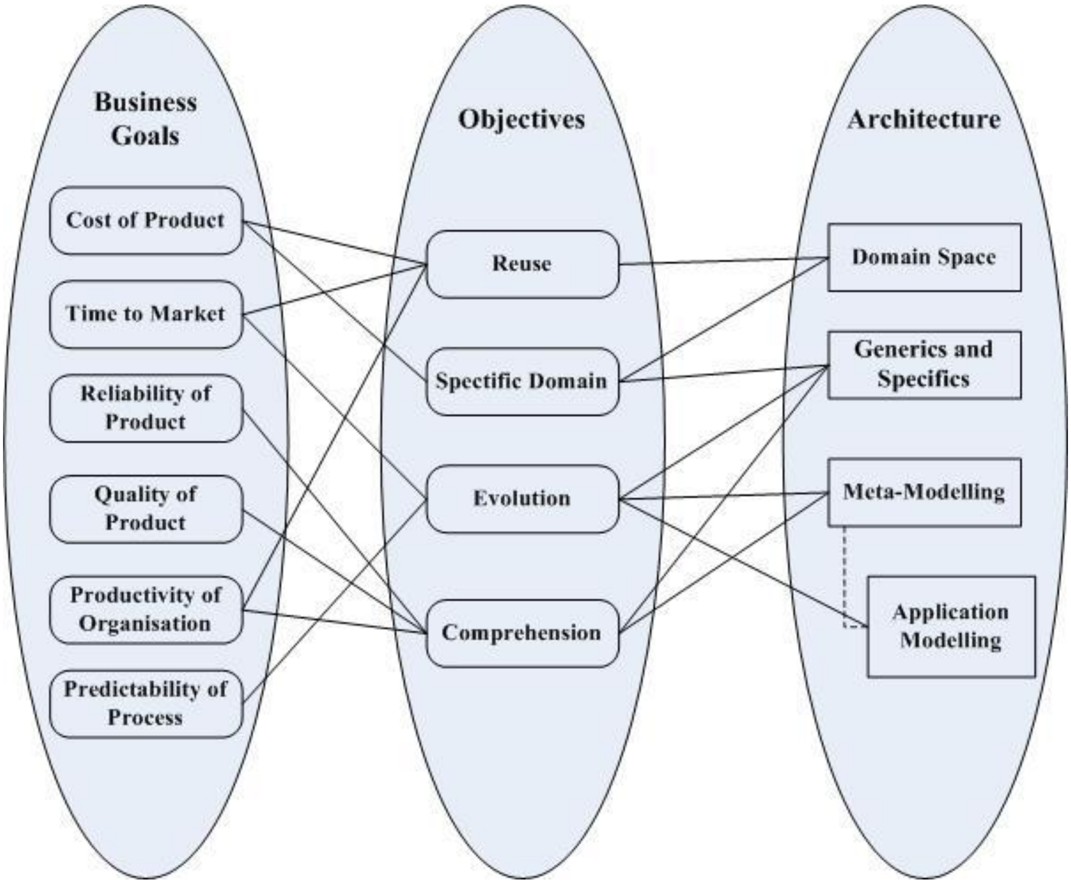


Figure 2. 1 Business Goals, System Objectives and Architecture

Although business goals are very general and hold for (almost) any business, it is obvious that some priority ordering is necessary per system (or market). Given the ordering of business goals, an ordering of system objectives can be derived, as illustrated in Figure 2.1. For example, the cost of product is related to the amount of reuse that can be established. Furthermore, system objectives can be mapped on software architecture. For example, when a specific domain is concerned it is good to explicitly distinguish generic and specific components.

2.3 Software Reuse

Software reuse is the use of existing software or software knowledge to construct new software. Reusable assets can be either reusable software or software knowledge.

Reusability is a property of a software asset that indicates its probability of reuse [33]. The purpose of software reuse is to improve software quality and productivity.

Software reuse is of interest because people want to build systems that are bigger and more complex, more reliable, less expensive and that are delivered on time. They have found traditional software engineering methods inadequate, and feel that software reuse can provide a better way of doing software engineering.

An important approach to reuse and one tightly coupled to the domain engineering process is generative reuse. Generative reuse is done by encoding domain knowledge and relevant system building knowledge into a domain specific application generator. New systems in the domain are created by writing specifications for them in a domain specific specification language. The generator then translates the specification into code for the new system in a target language. The generation process can be completely automated, or may require manual intervention.

Important contributions to generative reuse include the development of the theory of meta-compilers, also known as application generator generators. These tools assist in the development of domain specific application generators.

An important part of making domain engineering repeatable is a clear mapping between the outputs of domain analysis and the inputs required to build application generators. Better integration of these two phases of domain engineering will mean much improved environments for domain engineering.

2.4 Domain Engineering

The life span of the average software application is ten years with a large variance. Small-scale systems normally have a relatively shorter life. The type of application is a consideration in the expected life span. Administrative applications such as personnel and accounting systems live longer than business supporting systems for sales or manufacturing [118]. Thus, within the context of domain specific frameworks, an organisation must consider the scope, nature, and stability of the domain in order to determine the requirements for the reuse investment [3].

From the perspective of analysis, a domain is a well-defined set of characteristics that accurately, narrowly, and completely describes a family of problems. With respect to software development, a domain is a collection of current and future applications that shares this set of common characteristics [16].

Domain analysis is the key to reusable software in that it stresses the reusability of analysis and design over code [82]. A domain analysis identifies common architectures, reusable components, design alternatives, and domain-oriented terminology. It is expressed in terms of abstract classes and subclasses, protocols, frameworks, constraints, and inference rules, and finally encoded into design schemas, where appropriate domain-oriented terminology can be used to create application-oriented requirements language.

A domain model is the product of domain analysis. It provides a problem-oriented architecture for the application domain that reflects the similarities and variations of the members of the domain. An individual target system is created by selecting objects from the domain model to support the provided requirements. The domain model is also used to be the index into the object repository to ease selection and retrieval. New requirements or variations not present are flagged as unsatisfied. The proper function of the model is to capture how the designers and implementers think about the relationships among the parts of the system, and not necessarily how the relationships are implemented programmatically [13].

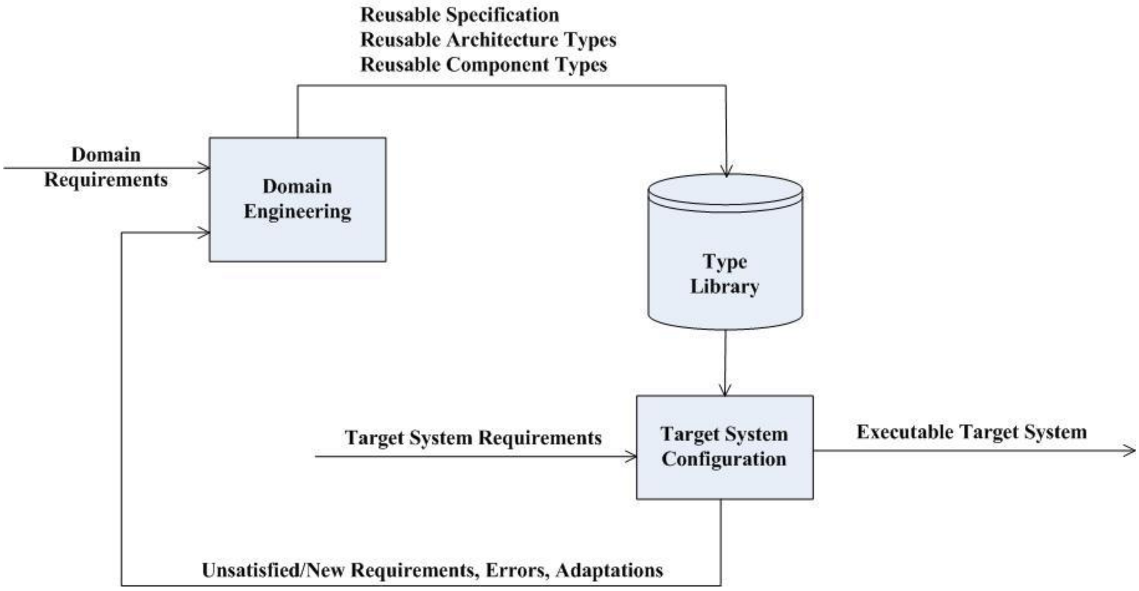

Reusable Specification
Reusable Architecture Types
Reusable Component Types
Domain Requirements
Domain Engineering
Type Library
Target System Requirements
Target System Configuration
Executable Target System
Unsatisfied/New Requirements, Errors, Adaptations

Figure 2. 2 Evolutionary Domain Life Cycle

EDLC (Evolutionary Domain Life Cycle) addresses the problem of software reuse. As shown in Figure 2.2, EDLC consists of Domain Engineering and Target System Configuration with emphasis on the production of domain reusable products. Domain engineering involves domain-oriented reuse – the combining of software development for reuse and software development with reuse. A cooperative effort between domain analysts and system analysts is required. Important aspects of the domain engineering are phenomenology, technology of description, and formalisation.

There is a diversity of opinion as to what products are the products of domain engineering. They range from creation of the domain model to complete application development. Specifications, designs, architectures [106], domain-specific code library [100], and DSLs (Domain-Specific Languages) and tools [44] are three such examples.

2.5 Executability of Model

Model describes system and its environment from a given view. It is an abstract representation of system and its environment. For a specific aim, model extracts a set of concepts relevant to the subject in order to make developers focusing on the whole system and ignoring irrelevant details [61].

2.5.1 Core Value of Model

In the information age, the software development uses the engineering method to build and maintain the effective, practical and high-quality software. The Software Engineering like the engineering method of other fields is all required to use models to participate in implementation of the project. The model has important practical significance for engineering method. In the building project, engineers make the building models to display and analyse the appearance and architecture of the building, in the project of aircraft manufacturing. The models are created by the engineers to carry out wind tunnel testing. In the engineering method, models can have a variety of forms, including the conceptual model -- for examples, the appearance of a building, the conceptual design of an aircraft; the graphical parametric model -- for examples, the architecture of a building, the framework of an aircraft; the physical model -- for examples, the miniature of a building, the proportional model of an aircraft.

The core values of models used in engineering method embodied in the following aspects:

- analysis and design of model-assisted: The model is a product of the physical analysis and design, but also the blueprint is mutually communicated by engineering staff of different roles, and it is usually produced by a minority of professional staff, the domain standard is used to describe the analysis of design thought of physicals, which can be a graphic or a character description.

- The cost of model is low and can be tested: Why do not make models the physical? Because the model is susceptible to change, while the cost of making model is less than the cost of making physicals. The blueprint can be given up easily, but the building cannot be given up. Besides, the model can be used in mutual communication of project-related staff, its core function is to be tested, in order to test it, and the available test standard is needed. If you cannot evaluate a model, the model is of no value.

- The model omits specific functions, but does not omit the details: The model often reflects one aspect of physicals. It omits specific functions beyond this aspect, but it cannot omit the details of this aspect. Models must depend on these details to verify its correctness.

Software engineering [105] uses engineering methods to build and maintain effective, practical and high-quality software. The model is the description and specification of software functions, structure, behaviour and its environment. The software model should have the core value of the model in the engineering method. It should not only stay in analysis and design of software, but also to be reflected in the correctness and effectiveness of the software validation, further to use the model to drive software development. In the BOF meeting of OOPSLA 2003, the experts defined the core value of a set of model-driven software development [120]. One of the most important things is to strive to achieve automatically built software according to domain model, to verify the software in developing is better than that of software requirements.

It can be affirmed that the core value of model in MDD is the executability of model. The executable models can be automatically transformed into system implementation; to validate the model by system implementation is the most direct and effective. The executable model includes analysis and design of the software, the cost of building an executable model is much lower than that of code implementation. At the same time, the executable models support reconstruction and omit the function realisation. But it does not omit the necessary details of system implementation. MDD is achieved by executable model, which is the key to dissolve "silver bullet".

2.5.2 Executable Model

Relative to the program executability, the executable models themselves is not executed in the computer environment. The executability of model is generally shown in two ways:

The first: models defined precisely can generate the executable code via the automatic and complete transformation process. The code is compiled into software system without manual intervention. Then it can be executed correctly in the computer environment.

The second: models are parsed as the operations with accurate semantics, and can be executed correctly in a specific software environment (such as the model virtual machine).

The model in the first form is an intermediate software product, which cannot be executed until it is fully transformed into executable code. The model itself in the second form is the executable software product, but it needs a specific execution environment to support. From the two forms, it is found that the executable models must have two necessary conditions: Firstly, the model is given with the executable semantics, and can be mapped to the executable code or operations directly. Secondly, the model execution is an automatic and complete process no matter in either way, the transformation of the model itself and parsing of model do not need the human intervention. The executability of model is the core of model validation and the core value of MDD. Thereby the software development can be driven by the model-centric method. The designers no longer need to care about the details of system implementation, an executable model eliminates the gap between the model and system implementation.

2.5.3 Modelling Maturity Levels

During the software development process, with the deepening of analysis and design of the systems, the corresponding system model will be experienced a refinement process from vague to the fine, from the simple concept of system to the structure and behaviour of system model. At the same time, in the analysis and design of complex systems, there are different model descriptions related to different stages and different abstraction levels, both a descriptive model making the simple natural language as a subject and a precise behaviour description model. The software is a complex man-made thing, the corresponding software model is a complex system, founding hierarchy and using it are fundamentals to analyse and construct the model. The model as a carrier to understand the behaviour of the system, essentially, the model not only has architecture, and also has a clear hierarchical structure.

To evaluate the description capacity and the abstract differences of models, MMLs (Modelling Maturity Levels) [22] is introduced and shown in Figure 2.3. The model hierarchy is divided into six levels (Level 0 -- Level 5) by the MMLs.

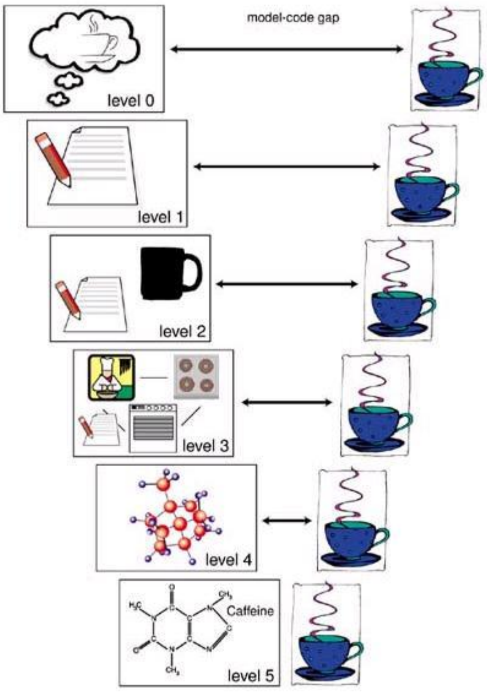


Figure 2. 3 Modelling Maturity Levels

- Level 0: No Specification. The specification and idea of software just exist in the minds of developers; there is no any model entity. This usually occurs in early software development, only a subjective idea of the system exists at the phase.

- Level 1: Textual. The specification and description of the software are expressed by text documents. The text document can be both purely natural language and a certain formalised document. As the subject is the natural language with the ambiguity, so its description and instructions of the software are quite vague.

- Level 2: Text with Diagrams. The specification and introduction of the software are expressed by the formalised document with descriptive diagram. The choice of diagram is freer, which can be any diagram describing system. At the same time, the adding of diagram makes the system easier to understand.

- Level 3: Models with Text. The model constructed by the diagram described by the modelling language. The system is described by model and natural language or the formalised document. Due to model-based and supported by formal modelling language, the specification and introduction of the software is more accurate and easy to understand and exchange.

- Level 4: Precise models. The software structure, functions, behaviours and environment can be represented accurately with the model which is described by the consistent and coherent formal notations or the modelling language. There is no ambiguity on software if it is defined accurately. And it can be mapped to the code directly. But it may be not complete.

- Level 5: Models only. The models constructed by the modelling language can completely, consistently and accurately describe the system in detail. It is sufficient to complete the work of code generation without the need for human intervention to realise the entire software system.

From the different levels of model maturity, it can be known that the distance between model and system implementation is closer and closer as the level increasing. There are too many non-formal descriptions of the model in MMLs 0 -- MMLs 3 level. The model itself is of ambiguity. Its abstract level is high and inaccurate. Basically, it is difficult to transform models into system implementation directly. The model only can be used as the model of analysis and design or the specification.

There is no ambiguity on the model of MMLs 4. MMLs 4 can describe the software specification accurately, and map the model to the code directly. But it may not be the complete code implementation，and need the manual supplement and improvement. MMLs 4 is still at a high abstract level. It also describes the system from various perspectives. But it may not be system-implementation-oriented completely.

The model of MMLs 5 can describes the system completely, consistently, in detail and accurately. And it can be transformed into the software application system completely and automatically. So the executable model is realised really.

The software developments in MDA focus on building a high-level general system model. The MMLs 4 model is the goal for MDA to achieve. At the same time, MDA is committed to making the model in MMLs 4 accurate and complete, and can automatically and completely complete the code transformation [22]. The designs of the model in the thesis mainly refer to MMLs 5 level standards, accurately and completely describe the system and construct executable models.

2.5.4 Transformation between Model and System Implementation

The ultimate goal of MDD is to make models transform into system implementation automatically, which is a necessary condition for the executable model. It involves two key elements: model and code generator. They depend on and restrict each other. The precise and complete model definition makes the size of model large and the relationship complex, but the complexity of code generator can be reduced; on the other hand, vague and incomplete model can be defined relatively simple, but code generator will become more complex, with a certain degree of adaptation and intelligence, as shown in Figure 2.4.

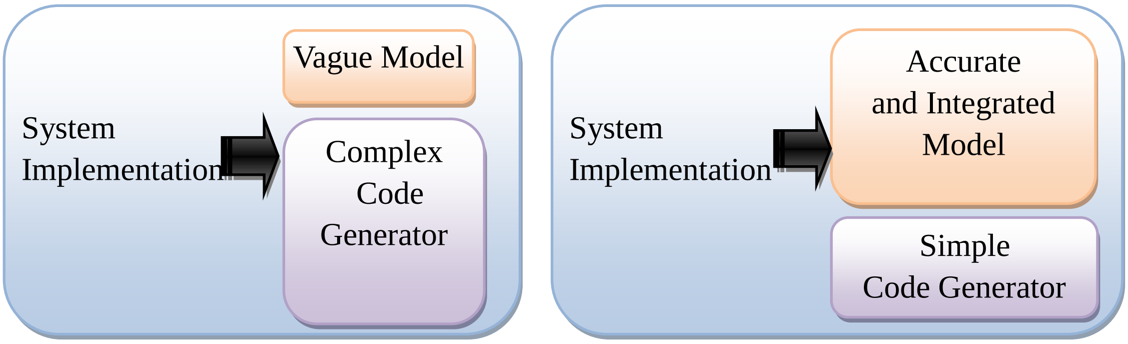


Figure 2. 4 Constraining Relationship between Model and Code Generator

The automatic transformation between models and system implementation can be achieved in two ways.

The first way is to refine models and reduce the degree of abstract, so that models can not only describe the system accurately and completely, but also gradually approach to the system implementation. The MMLs is a standard to measure the model description ability and the abstract degree. The refinement of model is also a process from vague to accurate, from abstract to concrete, to narrow the gap with the system implementation step by step, so the software system can be generated efficiently. MMLs 4 and MMLs 5 are the more suitable model maturity levels for code generation. For the model at MMLs 1 to MMLs 3, system implementation is difficult to achieve, or even impossible to achieve. The construction of related executable model based on UML is developed by following the thought, the architecture of UML has fundamentally changed starting from UML2.0, it emphasises more on behaviour modelling, and introduces some advanced language elements. At the same time, OCL is used to constrain the model accurately. In xUML (eXecutable Unified Modelling Language) and xtUML (eXecutable and Translatable Unified Modelling Language), the action specification language is directly used to describe the system in supplement, more accurate and complete close to implementation, and it is convenient to be transformed into system implementation.

The second way is to improve code generator technology and make code generation more adaptable and flexible, so as to gradually approach to the model description. MDD is built on existing technologies. Advances in computer technology will promote the realisation of MDD; the object-oriented technologies have contributed to the success of UML; the gradual development of computer technology, such as component technique, distributed technique, artificial intelligence and data mining technique will promote the realisation of code generator or a further model virtual machine, automatically and completely make the model transformed into the system implementation, and even the relatively vague software model.

2.5.5 Obstacles to the Executable Model

The goal of the executable model is to transform the model into the system implementation automatically. The model is an abstraction of high degree of system implementation, which simplifies the complexity and omits specific implementation details. Due to lack of the accurate description, as the model is transformed into a system implementation, code generator is difficult to generate the full implementation code, only the traditional coding methods are adopted to realise software systems. The model itself and the corresponding code generator are the main obstacles for the executable model at this stage. For the model:

- The description of model is inaccurate; the model itself is highly abstract, omitting many details of the definition, resulting in the semantics of most of model elements inaccurate, with a lot of uncertainty and ambiguity.

- The description of model is incomplete. The model is based on a perspective to look at software, with a certain one-sided; each model scenarios is difficult to be combined to describe the whole system;

- The description of model is more likely to describe the system structure. The capacity for describing system behaviours is weak. Software itself is dynamic, so does the software requirements specification. A large number of static models are not sufficient to describe the software system;

For code generator:

- Code generator not only contains core domain business code, but also adds adaptation conditions and transformation logic according to specific modelling language. If model is incomplete and inaccurate, it even needs to carry out some logic judgement based on the external environment of model elements or relationships, or using scenarios, so as to generate the code automatically. Thus the complexity of code generator, system size and difficulty of implementation far surpass the generated system;

- Code generator is also a software product. Changes for software systems come anytime and anywhere. The complex code generators also need to face requirement changes, not only for the generating system, but also its own changes. The excessive complexity and the huge scale of the system will make code generator difficult to cope with requirement changes;

- Code generator according to transformation logic and model semantic to carry out code generation, code generator is often associated with the modelling language. Abstract syntax elements can be gotten from the model instance and according to its semantics to be transformed into a specific executable code. As the modeller cannot clearly understand the semantics of the model or semantic expressed by the model element itself is not clear, the model constructed by the modeller cannot be generated the correct code by code generator.

Many issues between model and code generator restrict the executability of the model, so the efficiency of code generation exists in automatic generation platform, namely, the code framework can be only generated in accordance with the model definition. The auto-generated code cannot fully meet the design requirements of software functionality, it is still needed to manually add some code, or connect the code fragments. However, with the development of techniques and the gradual refinement of model definition, the executable model will settle the obstacles.

2.6 Summary

In this chapter, the background and basic concepts of the thesis are introduced.

- Software engineering is the engineering application of software development, operation, and maintenance. The development models in software engineering evolved from prototyping to MDD. MDD is the target of the thesis.

- The concepts and elements of software architecture, software reuse and domain engineering are introduced. They are the background of the thesis.

- The executability of model is the core value of model in MDD, which results in the main idea of the thesis. The executability of different models is investigated and MMLs 5 standard is used as the guidance for the thesis work. How to make model executable and the obstacles to the executable model are discussed in this chapter to do the preparation for the next work.

Chapter 3

Related Research

Many organisations, companies and research institutes are carrying out an active exploration and research about model-driven development methods, modelling methods and implementation techniques, as well as executable models. OMG puts all the explorations together and MDA is formed, it also makes some standards to support MDA, such as UML, MOF, OCL, XMI, CWM, QVT, etc. However, the biggest problem of OMG is that it elicits the whole architecture, but does not provide the concrete implementation [55].

The general modelling tool supporting UML is provided by IBM, such as Rational Rose, but it has some problems in some aspect, for example, Rational Rose is not fit for domain modelling [115].

This chapter introduces and discusses the background and the state of the art of the related fields including MDD, MDA system and the executability of MDA, DSM, web services and web service composition techniques, etc.

3.1 Model Driven Development

MDD (Model Driven Development) is a new software engineering method which is developed following the object-oriented development methods. It focuses on system modelling based on the best practices to construct software system models. Models are used to guide requirements analysis, system design, code design, system test, and system maintenance at various phases of software development. MDD involves some technical methods, such as model description, modelling methods, model transform and code generation, etc.

MDD is a model-centric software development process, and the model itself can have many forms, but an accurate language is needed to be defined to describe the system or part of the system. The model is the description of the system (or part of the system) by a precisely defined language. The description language has the precise form definition (syntax) and the meaning (semantics) definition. Such a language is suitable for computer to interpret automatically [56]. The underlying purpose of MDD is to make model and implementation be unified perfectly. The models are transformed into system realisation by modelling, model transformation and code generation. At the same time, the models can be used to answer the requirement changes rapidly. The high flexibility of the models decides that it is only needed to adjust the models and re-generate the code, which which is better for responding to requirement changes.

MDD brings reform of the system development, which improves efficiency of software development and enhances the portability of the software, ability of team work and maintainability. MDD improves the abstract level of development; the modelling is carried out above the code realisation in a manner of code generation to make the highly efficient and stable system come true, which greatly improves software productivity and reliability.

However, MDD just gives us specification and methods of development; it is not the real problem-solving entity. It requires a powerful tool to support, including modelling tool, model transformation tool as well as code generation, etc. while MDD does not abandon the existing software development methods and techniques; it just takes a solid step on the forward road of software development method. Due to these advances in software development methodology, so the successful application of

MDD approach becomes possible, for examples, 3GL [22]，design patterns [111], component-based development [125], middleware [11], declarative specification [97], application framework [26], design by contract [72] and the object-oriented development methods [40], etc.

3.1.1 Model Driven Architecture

MDA is a software development framework defined by OMG, which is based on UML, MOF, XMI, CWM, and CORBA. It supports software design and model visualisation, storage and exchange. MDA separates the tightly coupled relationship between analysis and design of business function and implementation techniques. MDA development process is actually a process of model-centric, changing the high-level abstraction model into low-level ones, which is finally transformed into code.

MDA is put forward by OMG (Object Management Group) in 2001, and it is an essential change from object-oriented design to model driven development [80]. Its core idea is to abstract the core PIM (Platform Independent Model), which can completely describe the business function and have nothing to do with implementation techniques, then multiple transformation rules are made according to different implementation techniques, and PIM is transformed into PSM (Platform Specific Model) by these conservation rules and assistant tools, PSM have some with the implementation techniques, finally the enriched PSM is transformed into code. The purpose of MDA is to separate business modelling from underlying platform techniques by PIM and PSM and to protect the modelling result that cannot be affected by technical change [21].

The essence of development approach based on MDA has raised the role and status of model in software development and the model-centric idea is used to drive the entire development process, namely, the model is used to guide the understanding of the software, design, create, deploy, operate, maintain and modification[33]. MDA can be used to answer the challenges of interoperability. It is an open and vendor-neutral development method. It is built upon the existing OMG modelling standards, and takes full advantage of value of these existing standards. MDA system architecture is shown as Figure 3.1.

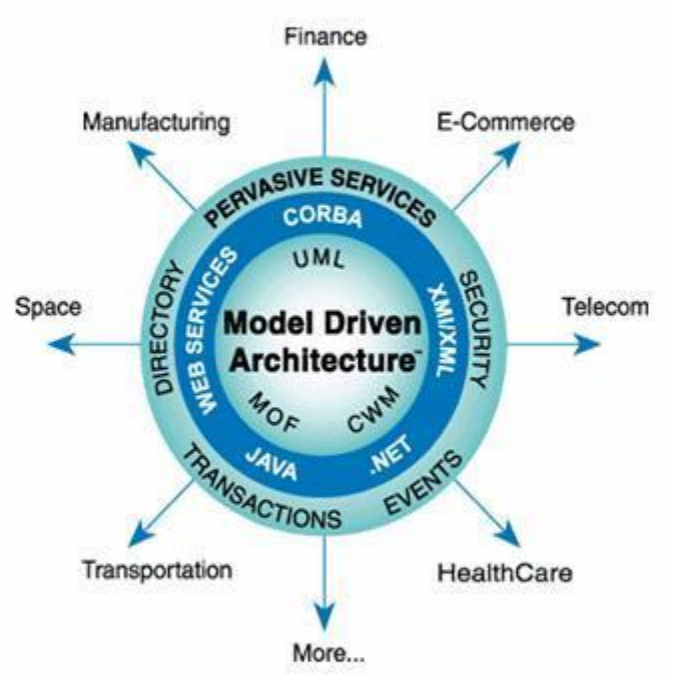


Figure 3. 1 MDA System Architecture

The systems development process based on MDA is with the following four characteristics [109]:

- The development process is completed by concept models of different abstract level and many viewpoints.
- It makes a clear distinguishing between PIM and PSM.
- The model plays an important role not only in the initial stages of development, but also in maintenance, reuse and further development process.

- The model records the relationships among different models; therefore, it provides the basis for model refinement and transformation.

3.1.2 Key Techniques of MDA

The core techniques of MDA include UML, MOF, QVT, CWM, XMI and OCL.

MOF (Meta Object Facility) [86] is a language used to define modelling language to provide support for a wider range of applications. MOF provides a unified way to describe the different types of modelling structures. So, a unified approach can be used to describe properties of model structure that make up of the model and the relationship between the model structures. MOF is the core technique of MDA.

UML (Unified Modelling Language) [91] is a standard modelling language to use MOF to define meta-model, and MOF can be applied to almost all applications and platforms. UML is the basis for the existence of MDA, and all the applications created by MDA techniques are based on a standardised, platform-independent UML model. UML is used by MDA to describe a variety of models, but it is not for MDA. However, as the most popular modelling language of current, UML has occupied 90% market share of modelling language of the world. It becomes the de facto standard of modelling language [103], which is the basis of MDA and is also the most powerful weapon of MDA.

XMI (XML metadata Interchange) [93] is meta-data exchange based on XML (eXtensible Markup Language), which aims to facilitate exchange between the data of UML modelling tools and metadata, and provides a metadata storage mechanism in a multi-tier distributed environment. It defines data exchange format based on XML for various models by standard XML document format and DTD (Document Type Definitions) [96]. This makes the models as an ending product can be transferred in a variety of tools to ensure that MDA will not be added a new layer of constraints after breaking a restriction. XMI specification supports any data transformation of meta-data (including model and meta-model) that can be expressed in MOF. At the same time, it still supports the transformation of the entire model or a fragment of the model to XML.

CWM (Common Warehouse Meta-model) [89] provides a means of data format transformation. At any level of model, CWM can be used to describe the mapping rules of two kinds of data models, such as transforming the data entities into XML format from relational database. Under MOF framework, CWM is likely to make a common data model transformed into engine.

QVT (Query/View/Transformation) [37] are the new standards of OMG, which is being developed, mainly to solve the problems of transformation realisation of model. MOF is used to define QVT, which is a part of MOF.

OCL (Object Constraint Language) [87] is an indispensable part of MDA techniques. It can be used to constraints model at any level of MOF four-layer models and instances. Its real meaning lies in the modelling-related domain constraints language, in addition to constraints model, an important usage of OCL is to describe the model transformation rules.

3.1.3 Hierarchy of MDA

The model is the focus of attention of MDA, from the practical perspective, model is abstraction of software entities of different views during the software development process, and at the same time, it guides software development. MDA divides model and the meta-model into four layers [71], as shown in Figure 3.2.

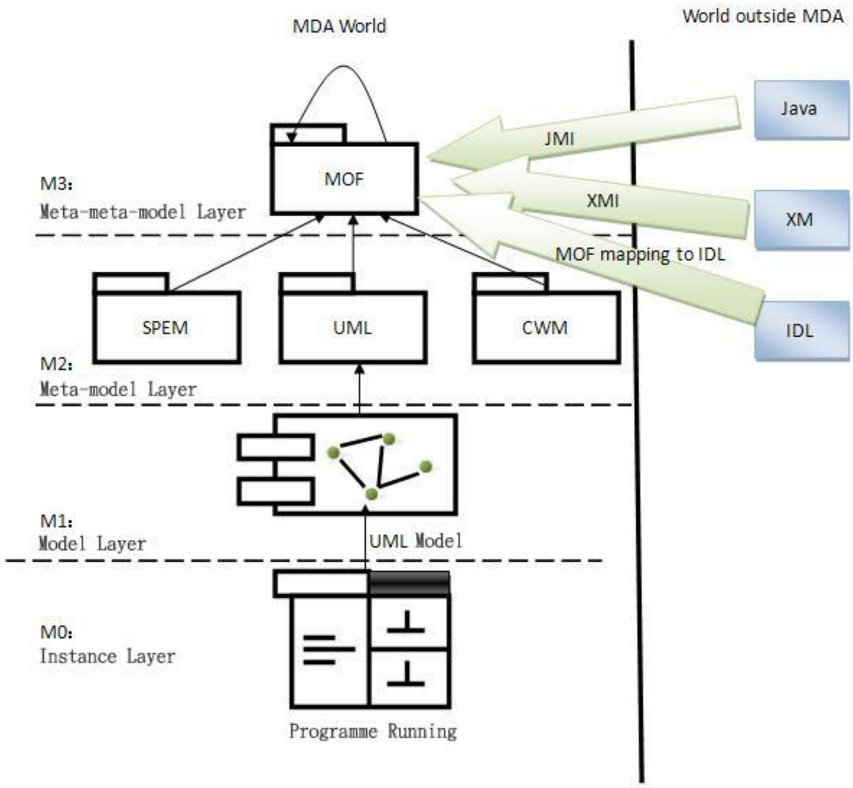


Figure 3. 2 Hierarchical Model in MDA

The M0 layer is the instance layer, the running system is at the M0 layer, at which is instance. On the point of business modelling, the instances of M0 layer is the business object. For Example, data in the database or running active object in computer.

The M1 layer is the model layer, including models; the concepts of M1 layer are the classification of the instances of the M0 layer. Models are usually faced by the modeller, such as UML model.

The M2 layer is called meta-model layer, which is corresponded to meta-model M1-layer. The meta-model of M2-layer extracts abstract concepts and relationships structure of different domains, and provides a modelling notation for the M1-layer modelling. That is, the M2 layer provides domain modelling language for different domains.

The M3 layer is meta-meta model layer, and MOF is located at the layer. MOF provides a more abstract level of modelling support which is needed by defining the meta-model at the M2 layer. MOF is the meta-model of all the meta-models of the M2 layer. At the same time, it is self-describing. MOF can be used to describe MOF meta-model itself. In MDA framework, the M3 layer only has a model -- MOF, which is the most basic and core standards of MDA, and it provides a unified semantic basis for all the models and the meta-model in MDA, making a unified model operation based on MOF become possible.

3.2 Executability of MDA

The executability of model of MDA is reflected in transformation from PIM described by UML to PSM, and then PSM is transformed into executable code, so the model is transformed into executable code. The model transformation rules and precise definition of PIM are necessary conditions for MDA to make model transformed into system implementation, while the key is the accurate and complete definition of PIM, because no matter how subtle model transformation method is, it is not able to complement deficiencies in the model itself.

UML is a well-defined, easy to express, powerful and general modelling language, which is used as a description language for PIM by MDA. PIM is a description of software features of platform independent and specification, the software features mainly include architectural feature of the system (static) and behaviour characteristics (dynamic). MDA demands high quality of PIM. PIM must ensure the completeness, consistency and unambiguous, otherwise, it cannot be used to generate PSM through the model transformation; neither can it be accurately and completely translated into a system implementation. UML can be better modelling structural features of the software, and PSM generated from PIM can carry more comprehensive structure information of the system, such as class diagrams, deployment diagrams, etc.

However, UML is not good at decrypting software behaviour features, although UML provides a sequence diagram, state diagrams, activity diagrams, collaboration diagrams and other model views to modelling the software behaviour features, the semantics of these model views and its model elements is inaccurate, which cannot provide the necessary details of system behaviour. Therefore, the quality of PIM which is solely described by UML is not high, PSM after model transforming cannot fully reflect for software information of platform-related, and the missing software behaviour features are needed to be manually added to PSM, so that PSM can be used for the code generation. In order to improve this defect, making UML completely and accurately describe PIM, in particular the behaviour features, the capabilities of UML to describe the software behaviour characteristics must be expanded. Therefore, UML is improved by OMG and UML 2.0 is released. Meanwhile, two solutions are provided based on the basis, one is to make combination of UML and OCL to describe PIM, the other is to use the executable UML to describe PIM, the most representative is xUML and xtUML.

3.2.1 Extension of UML 2.0

According to some problems existing in UML1.X, OMG releases a new UML 2.0 standard in 2003 [118]. UML 2.0 integrates action semantics [84], extends behaviour diagrams, adds loop, condition, assignment and other control and operational structures, and enhances the ability of profile to express dynamic behaviour [35]. A broad look at UML 2.0, it is not just a modelling language, but a combination that can be used to define meta-meta core of a language family and a meta-language of general-purpose modelling. As a narrow UML, compared to UML 2.0 with the previous version, it is greatly enhanced in the component-based software engineering, real-time and embedded systems, description ability of business process. The improvement of UML 2.0 mainly focuses on the basic structure and the upper structure.

The basic structure of UML 2.0 defines a core infrastructure library of meta-language, a self-shown UML meta-model can be defined through reuse of the core, and so do other meta-models, including MOF and CWM. Because they use the common core library, so UML, MOF, and CWM are more consistent in the architecture. At the same time, the infrastructure library also provides a more robust mechanism of customisation UML that allows user to define dialects according to different platforms and fields.

The superstructures of UML 2.0 strictly reuses constructs included in the Infrastructure, which improves support of component-based development and MDA, optimises ability of the structure specification, and enhances scalability, accuracy, integration of behaviour diagram. It is embodied in the following aspects [69]:

- Model Diagrams：UML 2.0 supports 13 kinds of diagrams, which can be divided into two categories: the structure diagrams and the behaviour diagrams. The former includes: class diagrams, the composite structure diagram, the component diagram, the deployment diagram, the object diagram and package diagrams. The latter includes: the activity diagram, the interactive diagram, the use case diagram and the state machine diagram, among them, the sequence diagram, the communication diagram, the diagram of interaction and preview and timing diagram are collectively called the interactive diagram. Compared with UML 1.X, composite structure diagrams, package diagrams, diagram of interaction and preview and timing diagram are the new diagrams. The original collaboration diagram is renamed as communication diagram, state diagram is changed its name as the state machine diagram. The original implementation diagram is cancelled.

- Components：UML 2.0 enhanced support for component-based software development. As a modularised part of the system, the behaviour and state of the internal element contained in the component is encapsulate by the component itself with interfaces, and the interface is used to define its behaviour for the external and it can be replaced in its environments. Components provide system functionality by assembling and connecting the interface between the collaboration components. In UML 1.X, the concepts of components is mainly used in the design phase of system implementation, while the 2.0 will make the component used in the entire life-cycle modelling, and finally it will be optimised in the deployment and run-time environment.

- Interactions: According to different interactive purposes, UML 2.0 can express interactions in several diagrams: the sequence diagram, the communication diagram, the diagram of interaction and the preview and timing diagram. Among them, stretching capacity of the sequence diagram in UML 2.0 has been significantly improved. The New interaction occurrence, combined fragment and interactive operator makes the complex control structure, such as selection, loop, parallel, orderly, and references can be expressed in the sequence diagram. The diagram of interactive and preview is one of the new interactive diagrams that describes the interaction, particularly focuses on control flow, removes the message and lifeline, and the use notation of activity diagram. Meanwhile, the timing diagram is also a new, particularly suitable for interaction diagrams of real-time and embedded systems modelling.

- Activities/Actions: the activity diagram in UML 2.0 enhances the modelling capacity of complex process, supporting model of control flow as well as model of object flow, which enables the integration of the activities and actions; the activities define a flow diagram (process); the action defines the nodes of execution behaviour, making behaviour modelling more intuitive and effective. The core of its new constructs includes: the pin is used for input and output for the action, the structured nodes, and interruptible regions and so on.

Its semantic enrichment for the originally core constructs in UML1.X, which includes: adding parameters on the edge, such as flag, flow, and abnormal, etc., enhancing partition method for the activity diagram of multi-dimensional, hierarchical and expansion, and control node support bifurcation , convergence, decision-making, mergers and so on.

- State Machines: UML 2.0 realised complete encapsulation of sub-state machine and ability of pluggable replacement by exit / entry points on the border. The state machine can be specialised; a specialised state machine is an expansion of generalised state machine. At the same time, the sequence of Constraint Operation occurred in port or interface of component can be effectively constrained by the protocol state machine.

The making of UML 2.0 makes us see the dawn of MDD again. However， UML 2.0 still has not changed the fact of separation of MDA model and the system realisation so far. The main reasons are as follows:

Firstly, UML does not fundamentally change its structure, and it still uses structured abstract syntax to define the model and its elements, although the action semantics are substantially increased, it is still hard to be dynamically associated with true system and there are still shortcomings in detail description.

Secondly, UML is still a general modelling language according to all the fields, UML 2.0 make the whole system more substantial and hard to use, difficult to understand, more difficult to be transformed into system implementation.

Thirdly, the appearance of UML 2.0 did not change the missing problems in the model-implementation supporting environment.

3.2.2 Combination of UML and OCL

The semantics definition made by the meta-model is informal and half-baked, and it does not give precise constraints for the detail definition of semantics within the model. OCL enhances the semantics description ability expressed by the modelling elements based on UML semantic. The constraints are defined as follows by UML 2.0 specification; namely, it is a semantic condition or restriction. It is one of the three kinds of UML extension mechanism (prototype, tagged values, constraints), and OCL is usually used to formally express the constraints [46].

OCL is an expression query language. It plays a role in object model but does not change the state of the model. It has two central roles in the model semantic constraints, as well as the model query [124]. As a part of UML standard, which is used to describe an additional constraint relationship in UML model, such as invariants, pre/ post-conditions of operation, the state threshold and rules of attribute derivation, accordingly, it can further accurately describe UML model. OCL has the following characteristics [123]:

- OCL can not only be used to develop constraints, but also return the result from expressions defined as the model elements. Thus extends the scope of its application is extended.

- OCL has a good mathematical background. It is based on set theory and predicate logic, so it can accurately and unambiguously describe model elements.

- OCL is a declaratory, not imperative language. It is a language, which describes what to do and does not describe how to do. It is a query language, and its action does not make an impact or change on the model itself, which also means that the system will not be changed because of OCL expressions.

- OCL is a strongly typed language, and any element is type-related, the type of return value of any expression are certain.

OCL is a necessary condition for MDA, but not sufficient one. The typical development process of MDA is to establish PIM, describe transformation definition, and transform PIM into PSM by transformation tools. PIM and PSM are defined by the modelling language during the period, the role of OCL at that time as shown in Figure 3.3.

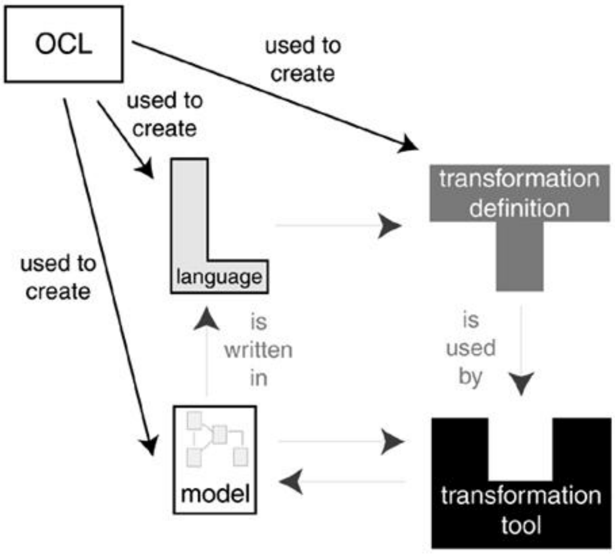


Figure 3. 3 The Role of OCL in MDA

The usage of OCL during the typical MDA development process includes three aspects [22]:

- The first is that OCL is used to describe constraints in the model on creation of models. To describe constraints in the instance of UML model is the most common application, the model view and OCL expressions are necessary for a complete PIM model, or the model cannot be precisely described. Only model constraints are made in detail, the automatic transformation of models is possible. According to the modelling, there are three types of constraints. The first is the invariant, which is used to describe the static structure constraints of the system. The second is the pre-and post-conditions, describing those conditions and constraints that must be met by an operation at the beginning of the implementation or at the end of execution. The third is the state threshold, which is used to express the admissibility constraints on state transitions in the state machine diagram.

- The second is that the definition of model transformation can be described by OCL. The corresponding rules between the meta-class described by the source language and those described by the target language are needed to establish to define model transformation. OCL expression is used to accurately determine the model elements of source and target meta-model, as well as their transformation rules. The source model is transformed into target model by the analytical implementation of the transformation tool.

- The third is that the modelling language can be described by OCL. Although OCL is called “Object Constraint Language”, actually it can be used to constrain the entire model in MDA four-layer model. The real meaning of OCL is to establish the related modelling constraint language. At the same time, it still can constrain specialised mechanism of UML profile, which is defined as a group of stereotypes, a group of related constraints and a set of tagged values. Using UML Profile needs additional syntax and mapping rules, the additional rules can be defined by using OCL.

OCL enhanced the accurate model description ability in MDA system and now quite a number of tools support OCL assistant-modelling. For example, a company named Klasse Objecten in the Netherlands released plug-in Octopus which supports Eclipse development environment of OCL2.0 [83]. The inspection tools of OCL Compiler 1.5 can be integrated into SELECT Enterprise and Rational Rose [48]. It not only enhanced the accurate description ability of UML, but also enhanced the constraint capacity of MDA four-layer model by introducing OCL, while by making the definition of rules of model transformation, making the model transformation of MDA becomes possible. However, if the current OCL to define model transformation and it is still needed to be extended, and then OCL did not change the nature shortcomings of UML, the model is still difficult to be transformed into the system implementation.

3.2.3 Executable UML

UML is a united symbols system, which is widely used to indicate various aspects of object-oriented symbols. UML is comprehensively used during the process of MDA, although UML specification is necessary in MDA process, it is inadequate to carry out an executable modelling. xUML is a subset of the executable UML. xUML abandoned the weaker semantics elements in UML, such as component diagrams and deployment diagrams, kept the strong ones, such as sequence diagrams, collaboration diagrams, state diagrams, class diagrams and package diagrams forming the core of a subset of UML, while it enhances the action semantics, and to establish an executable PIM based on this [108], as shown in Figure 3.4:

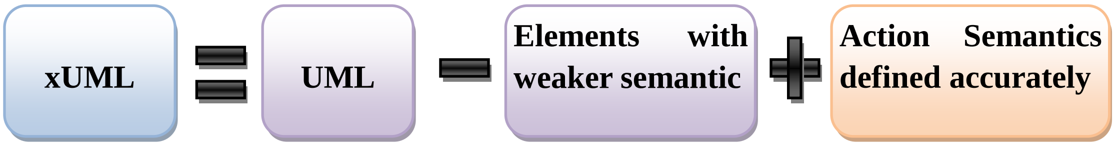

xUML
=
UML
−
Elements with weaker semantic
+
Action Semantics defined accurately

Figure 3. 4 The Basic Structure of xUML

The core of xUML is the accurate action semantics, the model based on object-oriented development methodology can be accurately described by the action semantics. At the same time, the transformation between xUML model and code can be realised by the mapping rules between model and code [75]. The behaviour semantic is designed to provide a way for the modellers to precisely define behaviour in UML model. The action semantic in coordinating with UML can accurately describe the model behaviour at the abstraction level above the programming language.

In xUML, the system is divided as follows: a domain: a subject business needing studying; categories: a collection of similar transaction; status: a situation of a class; operation: action of state-free. Generally speaking, xUML is the executable version of UML, which has a clearly defined and simple model structure, which contains the precise action semantics, an action specification language and a configured software process. Using an executable model-driven development has the following characteristics [75]:

- The accurate and complete analysis model: the analysis model is not related with the implementation, but the details. A PIM contains all the information belonged to the subject matter being considered, including the complex and analysis related detail part. When PIM gives the desired result of all the given test cases, this PIM is complete. The delivery of PIM is a result, not just the document. This requires that PIM is accurate, complete and executable.

- The scaled division method: it expresses the domain knowledge of xUML model in the domain specific subject business for the establishment of complex domain problems. Each subject business domain is called a domain, and the independent PIM is used to obtain and express information of each domain. A domain model encapsulates a subject matter, which can be a problem-oriented, and can be solution-oriented. The knowledge of a domain is described in the way of PIM, while PIM itself is executable and testing;

- An unambiguous standard symbol: xUML provides a simple, coherent subset of UML notation. The choice of these symbols is based on the structure of practical application, rather than special cases used in a construction of a software system. These symbols themselves are simple, and the way of organising and integrating those symbols must follow strict rules, which can maintain the clarity of the entire system specifications.

- A consistent process on concepts: xUML based on the model-driven development process is with consistency and strictness. Analysis is the process to understand all the subject business. Design is the subject matter of analysis. Developers generate the appropriate development products at an appropriate abstract level. These products will always keep the latest. Even after code generation, all the work is still carried out in the model.

- A large-scale reuse: in xUML, the domain is used for re-use. A domain represents a subject matter, and the domain can be mapped to software unit at the appropriate time, which encapsulates a subject matter or an aspect of the system. A subject matter can be reused in a large-scale by the loose coupling and cohesive of a domain.

Although xUML includes the precise definition of the action semantics, it does not make a definition of a specific action language, which is completed by the software vendors. The more well-known is ASL (Action Specification Language) [126] issued by a company named Kennedy Carter. It is a behaviour language that is independent of implementation language platform, by which the model behaviour description can be improved and an executable model can be established, too. The ASL is an unambiguous, accurate, readable and executable process based on object-oriented modelling techniques. The others are such as OAL (Object Action Language) [42], SMALL (Shlaer-Mellor Action Language) [76], TALL (That Action Language). At the same time, xUML supporting tools have appeared based on xUML and different action semantics. For example, the Products-iUMLite [113] of the Kennedy Carter supports development process of xUML, establishment and verification of executable models and the code generation. BridgePoint of Project Technology Company uses the OAL as the behaviour language supporting MDD of xUML. At present, xUML is mainly used in real-time systems development and the scope of application is relatively narrow.

3.2.4 Executable and Translatable UML

xtUML [95] is used as a subset of UML, and it integrates a complete object-action language. The Developers can create an executable domain model by this, after using xtUML modelling; the verification system will be carried out to verify whether the models meet the critical requirements. The validated model was compiled into platform-specific code, finally formed into the target system can be deployed.

xtUML separate analysis model and design of software, allowing the developers to be detached from target platform to test the analysis model of software, while application model compiler automatically generates a source code of target- specific platform and language optimised from the tested analysis model. xtUML is a well-defined and full automatic software development methodology based on UML notation. xtUML can accelerate development process of real-time embedded and industrial software project.

To surround the motives of completely isolating the application model and the design of software architecture, the design of xtUML includes the following three components: the application model (namely, software analysis model) to realise a clear and accurate modelling of software functions. The application model is executable, so it can be used to verify the functional requirements of software. The application model is completely independent of software design and implementation details; the software architecture (defined as a collection of design patterns, design rules and implementation techniques) is integrated into the translator, acting as a reference template generated by target code. The software architecture is entirely independent of the type of applications they support; the translator maps the application model to design rules and patterns corresponding to software architecture to achieve the full code generation automatically, as shown in Figure 3.5.

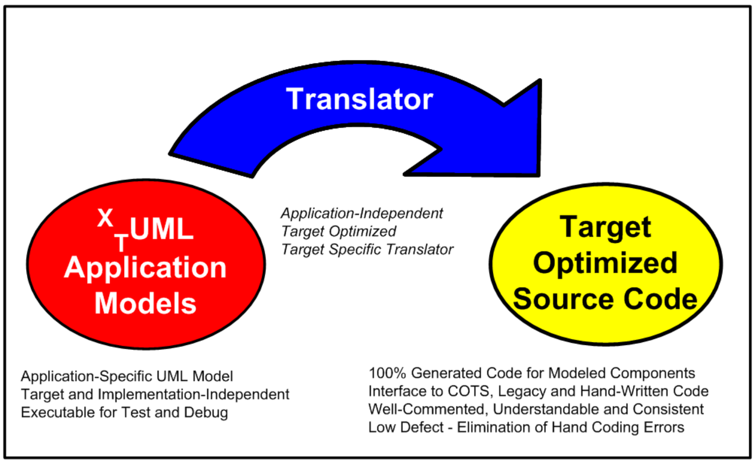

Translator
xTUML Application Models
Application-Independent
Target Optimized
Target Specific Translator
Target Optimized Source Code
Application-Specific UML Model
Target and Implementation-Independent
Executable for Test and Debug
100% Generated Code for Modeled Components
Interface to COTS, Legacy and Hand-Written Code
Well-Commented, Understandable and Consistent
Low Defect - Elimination of Hand Coding Errors

Figure 3. 5 The Development Model of xtUML

xtUML automatically generated the complete source code from the application model. The complete source code is optimised based on the target platform. Of these, the translator is the core of xtUML, and it is composed by following three parts.

- Software design elements for the translation is a collection of design pattern and translation rules, the translation rules (also known as translation prototype-Archetypes) provide the design patterns needed by the code construction, as well as when and how to apply or fill these patterns.

- Translation engine extracted information of xtUML application model, which explains the design patterns and translate rules, map model components to the design pattern and eventually generate a full source code.

- The run-time library provides target code modules obtained from a series of pre-compiled routines supporting translation.

The module syncopations of translator is helpful to customisation, construction and maintenance of the translator, the addition and modification of design patterns, transition rules and the run-time library do not need to modify the rendering engine. The code generation process controlled by the translator involves three steps: (1) the translator extracts required information from xtUML application model; (2) the translator choose a suitable design pattern for the model component to be transformed under the transition rule; (3) the translator uses information extracted from application model to fill design patterns and get the full source code. The effectiveness of this simple process lies in a filling of design pattern will usually lead to the filling and invocation of other design patterns or rules, and thus making the translation of a model components trigger nested translation of multiple components, while everything is done automatically by the translator [117].

Currently, xtUML is mainly used in real-time systems development, and its compiler uses a special code template language, primarily for the design pattern of a specific framework and application, it is limited in application scope, and almost unable to interoperability between different tools.

3.3 Domain-Specific Modelling

According to Capers Jones's software productivity research [52], the 3GLs increased developer productivity by an astonishing 450%. After that, the later introduction of object-oriented languages did not make the improvement much further. From the pragmatic perspective, the emergence of DSM (Domain-Specific Modelling) narrows down the abstract distance between domain concepts and its implementation, thus significantly improves software productivity, as shown in Figure 3.6.

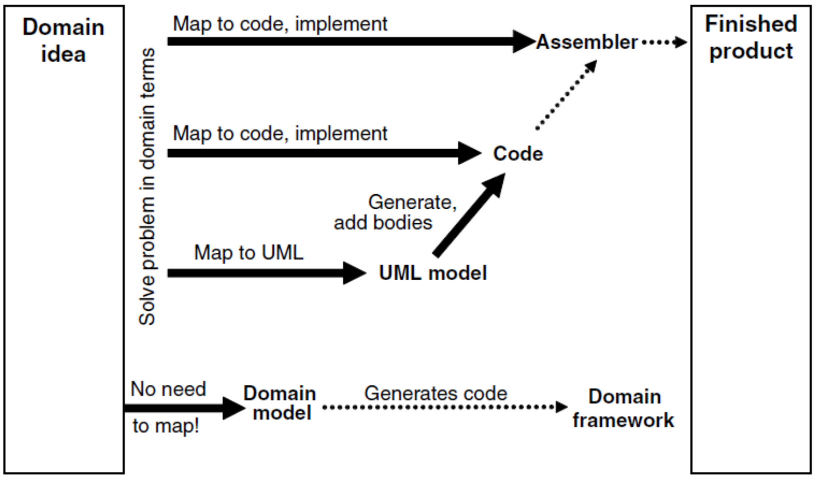


Figure 3. 6 Domain Concept Transforms into System Implementation

In the era of assembly language, developers use assembler to express *domain concepts*, to the period of the advanced language, developers use the advanced programming language concepts to represent *domain concepts*, in particular, the object-oriented programming languages make programmer able to more directly reflect the original appearance of the problem domain. The code written by the advanced programming language can be automatically transformed into assembler to implement the development of application system, and get the final product. At present, developers usually map *domain concepts* into a visual UML model, under the premise of generating part of advanced language code, add the missing code by hand and compile to generate the assembly code, thus final product is formed. From the assembler to UML model, the gap between the domain concept of human consciousness and computer realisation is getting smaller and smaller, but it is still great.

In order to further narrow down the gap between domain concepts and their realisation, DSM uses the domain model to represent the *domain concepts*. The information needed by the code generation is contained in the domain model, so the code can be generated automatically from the domain model, in the effect of domain framework, these codes turns finally into product. The way of the DSM hiding code is the same as the compiler hided in today programming way.

DSM mainly aims to do two things. First, raise the level of abstraction beyond programming by specifying the solution in a language that directly uses concepts and rules from a specific problem domain. Second, generate final products in a chosen programming language or other form from these high level specifications. The modelling language, code generation and framework code are required to be applicable for the requirement of a specific application domain, so the automation of application development becomes possible. In other words, the application development is domain-oriented and under the users' controls completely [53].

3.3.1 Architecture of DSM

DSM brings such a benefit: modellers only have to focus on the using of domain concepts to design solutions, rather than on the software architecture and programming details [20]. Once the design of solution is completed, it can be directly generated into code under the effect of code generator [43, 54, 25], making developers freed from burdensome code writing, thus the productivity of software is further improved, industrial experience shows that productivity is creased by 3-5 times [19, 64]. In addition, code generator is designed by domain experts, so the quality of generated code is higher than code written by the common programmer [19]. DSM can effectively and as early as possible find out the problems related to domain business and may appear in the modelling process, and to solve them by adjusting the domain model [133]. DSM puts forward a three-tiered architecture in the target environment [43]:

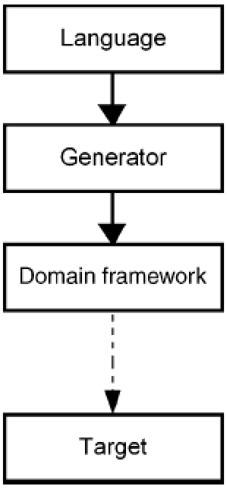


Figure 3. 7 DSM Architecture

1. Language

DSL (Domain-Specific Language) [18] provides an abstraction mechanism to deal with complexity of a specific domain. The cores of DSL are concepts and rules which represent things of application domain, rather than the concepts in a given programming language. In general, the main domain concepts are mapped to the object of modelling language, while other concepts are mapped to the object properties, association, sub-models or model links in other languages. DSL embodies business rules of a domain, such as operation specifications or industry standards. DSL makes developers feel the directly use the domain concepts to work [45].

DSL is defined as a meta-model supported by relevant notations and tools. The meta-model is the concept model of the DSL model, a model used to describe model. The meta-model describes the concept of DSL, nature, the legal association between the language elements, model hierarchy and correctness rules of model.

2. Generator

Generator specifies how to extract information from the model and to transform them into code. The simplest situation is that each modelling symbol generates some fixed code, including argument value input in the symbols. The Generator generates different codes according to values of different symbols, a different relationship between the symbols and other symbols, or other information in the model. The generated code will be linked to the framework, and compiled to generate the final executable code. The goal of creating a DSM solution is that there is no need to modify or extend the generated code manually.

In DSM, code generator interprets or compiles the model into executable code. By providing automatic transformation, code generator is helpful to the realisation of productivity and quality advocated by DSM. From the perspective of modellers, the generated code is full. It means that generated code is complete, executable and quality guaranteed. Namely, there is no need to manually rewrite the code or make operations on the code after code generation. Not all the code used in DSM is automatically generated, which is the reason for the existence of domain framework and target environment in DSM. They can be generated from different models or realised by manually programming. The generator itself, as the framework and target environment, is not visible for developers to a large degree. The invisible way is the same as black-box components or compiler that are not visible for developers. At present, code generators are mainly realised in three ways [27]: template, patterns and graph traversing.

3. Domain Framework

Domain framework provides interfaces between the generated code and the underlying platform. Usually, the framework code is not needed, generated code can directly call the components of underlying platform, and the existing services of the component are enough to support the execution of the generated code. For some complex domains, generated code is not executable alone, it cannot execute until cooperating with platform code provided by the framework at a target environment. At the same time, to define some additional and effective code or components makes the generated code easier. These codes of framework can be changed in size, from the component to programming language statements appeared in the chosen domain.

3.3.2 Domain-Specific Modelling Language

DSL ideas are proposed since the language of the first computer proposed [68]: In fact, DSML accelerated the early development of programming languages. Many years later, the thought that drives the development of modern languages remains exactly the same as the first computer language appears: improved the abstract issue allows the rapid creation and maintenance of a complex application [114]. DSL is a language which is designed to provide tailor-made symbols for an application domain, and it is only based on the concepts and characteristics of the domain. Similarly, DSL is means to describe and generate members of program family in a given domain, no requiring the programmer with a general programming knowledge. By providing special notations for the application domain, DSLs provide the substantial growth in productivity, and even make end users to design program become possible [58].

DSLs are defined by developers to solve domain specific problems. Martin Fowler believes that [32]: DSL is not a new concept, the early "Little Language" of Unix uses Lex and Yacc to generate code, as well as languages defined in the LISP are examples of using of DSL techniques. Karl Frank believes that [34], DSLs can be any language for a specific domain. In fact, DSL is a computer programming language used to solve problems of specific domain, which provides a suitable, fixed abstract concepts and symbols of the domain [18]. DSL is usually small, focusing on the statement rather than a plethora of rules or orders, and its expressive power is poorer than GPL (General Purpose Language). The expression of DSL can be plain text or graphic symbols. Since DSL deal with problems of a specific domain, the using objects are not only staff, but also domain experts, and even a domain specific grammar that is simpler and can be modified on end users' own can be defined. One of the DSL goals is that domain logic can be modified on the end users' own, while developers focus on developing the DSL support tools, rather than strive for the changing requirement. With DSLs, the majority of software requirements that are easy to change may be given the end users to control on their own and adjust the software. The more part can be controlled by DSL, the lower software maintenance costs are, and developers will be able to concentrate on other tasks of more valuable.

DSML (Domain-Specific Modelling Language) is a kind of DSLs. DSML is a new branch of modelling language, which is plotted out from the category of DSLs for emphasising model-driven design. DSML inherits the merits of DSL and has prominent features at the same time. The most important is that DSML has an ability of supporting meta-modelling. In the DSM methodology, the modelling work is actually divided into two parts, the first is to construct modelling according to domain concepts and rules that may exist in target application, namely, to establish domain meta-model; the second is to carry out domain application modelling on the target application system by using the result of meta-modelling (DSML). Among them, the capability of supporting meta-modelling is the core task of DSML.

3.4 Web Services and Web Service Composition

In recent years, web services become the main concepts of packaging and sharing of resources in an open network environment. Service providers can provide their own software to users in the form of web services; users can choose the wanted services in a distributed environment. At the same time, web services can simplify the complex software application styles and provide good support for resources sharing and cooperation work in the distributed environment by abstracting applications and resources at different levels into a unified form, and providing with them through the standard method. Web service provides a mechanism of description, management, sharing and services. It can make applications of different organisations in different regions and different businesses collaborated and interoperated effectively.

3.4.1 Web Services

Web service is an internet-based distributed component, which meets the service requirements of technique-neutral, loosely coupled, location transparent and can provide description, discovery and call for cross-enterprise applications [98]. The W3C gives a more accurate definition: web service is software application identified by URI (Uniform Resource Identifier), the application interfaces and bindings can be defined, described and discovered by XML product, at the same time, the application can directly interact with XML messaging protocol based on the Internet and other software applications [121]. Web services have the following characteristics [65, 29]:

- Intact encapsulation: web service is a service object deployed on the Internet, which has a good encapsulation, for users, who can only see the function list provided by the services, which is self-contained executable program unit and can provide specific services.

- Loosely coupling: The feature stems from the object and component techniques, when the realisation of web service changes, it does not affect service users. For the users, call interfaces of service providers remains unchanged, then any changes of the service implementation are transparent to them. The loosely coupling is greatly improved the flexibility of web service development.

- Self-Describing: web services explicitly describe its structure by using a computer-readable form. The Service description is intended to make users of the services can accurately understand the service and correctly use the service. The Self-description is the premise of loose coupling services, transparent location, but also provides technical assurance for the discoverable service.

- Interoperability: web service is platform-independent and language-independent, the services realised by different languages and platforms can communicate with each other. The interoperability is an important feature of service-oriented architecture, the core idea of services is to construct a general platform-independent, language-independent level based on various existing heterogeneous platforms, and applications beyond several different platforms rely on this layer to realise the mutual interaction and integration.

Web service is a self-contained, self-describing, modular web application that uses the standard XML messaging techniques to package information, and can access its interface through the network to accomplish a specific task. The separation of service realisation and service interface accelerates the applications based on web service into loosely coupled, component-oriented architecture [41]. The intensions of web services can be understood from the architecture and the protocol stack.

1. Architecture of Web Service

In 2001, IBM proposed a model which clearly describes the interactions between the various actors in the architecture of web service [59].

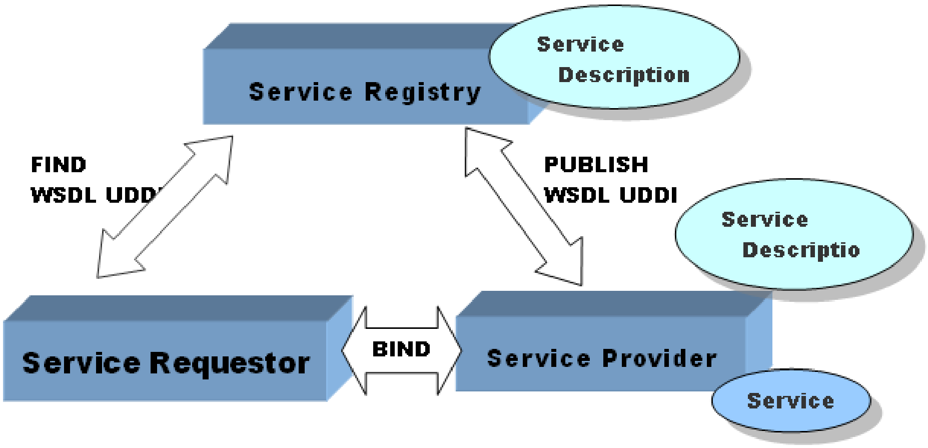


Figure 3. 8 Web Service Architecture

Components in the web service architecture must have one or more above-mentioned roles. The components in Service Oriented Architecture totally have three kinds of roles:

- Service Provider: release their own services, and response to the requests using their services.

- Service Registry: This is a searchable service descriptions registry center; service providers publish their services description here. During period of static binding development or dynamic binding execution, service requestor find services and get binding information of web services (in the service description).

- Service Requestor: This is a service consumer, using service broker to find the services they need, and then use the service.

Three kinds of operations used in these roles:

- The PUBLISH operation: Making Service Provider can register their functions and accessing interface to the Service Registry. Location of publishing service description can be changed according to the requirement of application.

- ➢ The FIND operation: Making Service Requestor can find specific types of services by service registry. In the finding operation, the service requestor directly retrieves the service description or queries the type of requested service in the service registry.

- ➢ The BIND operation: To enable service requestor can really use the service provider. In the binding operation, the service requestor uses binding details of service description to locate, contact and call services, thereby to call or start interaction with the service at run-time.

2. Protocol Stack of Web Service

The characteristics of web services using standard protocols is an important reason for the successful application of web services, web services are built on standard techniques and protocols, as shown in Figure 3.9.

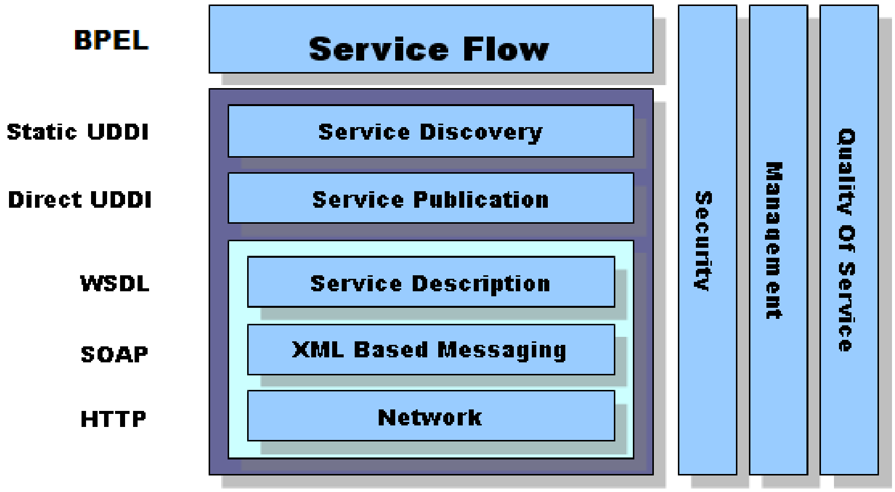


Figure 3. 9 Web Service Protocol Stack

XML (Extensible Markup Language) [85] is the basis for all the standards of web services. It is a meta-markup language used to describe the data organisation and arrangement structure in the data document, the importance of XML depends on standard-based, flexible, self-describing, extensible data format concept.

SOAP (Simple Object Access Protocol) [88] is a simple and lightweight XML-based web services exchange standard protocols. The actual goal of SOAP is simplicity and scalability. SOAP itself does not define any application semantics, by providing a modular package model and the data encoding mechanism to simply represent application semantics.

WSDL (Web Services Description Language) [92] uses the way no relying on any particular programming language and implementation methods to describe web service by using XML. WSDL defines a service interface for the service, as well as how to map interface to implementation details of the protocol message and the specific port address.

UDDI (Universal Description, Discovery and Integration) [119] universally describes, discovery and integrate standard, which is web service information registration norms used in a distributed network environment, but also is accessible realisation collection of the specification. UDDI is mainly composed by a business registry center and protocols accessing the center and API (Application Programming Interface). The core information model used by UDDI registration is defined by XML Schema [94].

BPEL (Business Process Execution Language) [1], web services are usually required to carry out reasonable composition in accordance with a certain granularity based on the specific application background and requirement to realise the full business logic. It is based on the way of orchestration or choreography to create two different aspects of the business process definition [99].

Web service is described by a standard language and published by the network, which can be discovered and called by software systems, with loosely coupling, reusable and interoperability features. After all, a single web service function is simple and limited, difficult to meet complex and volatile requirement of practical applications. Meanwhile, to maximise web service reuse, it needs using the existing web service as much as possible to reduce developments, and reusing high-quality web service composition and become the new powerful web service. With the extensive application of web services, web service composition techniques is also more and more widely concerned by the industrial and academic circles.

3.4.2 Web Service Composition

The composition of web service refers to a technique of selecting an existing, functional matching web service and combining them into a new service [8].The composition of web service is based on the dynamic characteristics of web services, including:

- Describable: can be described by the service description language.
- Redistributable: its descriptions information can be registered and published at the registration center.
- Searchable: the service meets the query parameters can be found by sending a query to the registration server and binding information of services can be gotten.
- Binding: the callable service instance or service agent can be generated by services description information.
- Callable: the remote call of service can be realised by using binding details of service description information.
- Composable: can be composed with other services to form a new service.

A composed web service is an aggregation of several mutually independent and interacted web services. The components are realised for itself by composed web services, sequential call them according to given combination models, and combine into the new web services which are more suitable for the requirement. From the structural perspective, the aggregation of web services is put forward after a higher level encapsulation of web service and treating the encapsulated function interface as web services. The new combined service is called composite service; web service used for composing composite service is elementary service.

Web service compositions can be divided into static composition and dynamic composition from the composition methods. The former makes the composition strategy between control flow and data flow of basic web services during the process of development and design; the latter one is dynamically going according to specific strategy in the system running, the composition if the control flow and data flow of basic web services is automatically generated by specific strategy. The dynamic composition based on the static one, but its difficulty is greater than the static one.

3.4.3 Modelling Methods for Web Service Composition

There are many formal modelling methods for web service composition. They respectively correspond to different description languages. Current research methods of web service composition can be divided into three modelling methods that are based on flow, cooperation and planning [67].

1. Web Service Composition Modelling Methods based on Flow

The composition method based on flow points out that compositive services are business flow built on a group of component services [15]. The web service composition method based flow uses the model which is similar to that used by the classic workflow modelling method to describe web service composition. Activity, control flow and data flow are the basic model elements of web service composition modelling. An activity corresponds to one operation of a component service. Control flow describes the dependency relationship among activities, which is the sequential relationship between operations of component services. Data flow describes the data transferring between activities, namely data exchange relationship between component services.

BPEL4WS is Business Process Execution Language for web services and shorted for BPEL, which is a description language of composition of web service put forward by IBM, Microsoft and BEA in 2002 [5]. BPEL combines web services by a process; the every step of the process is called an activity. At the same time, BPEL defines the atomic activities and control flow of structured activities, defines a partner and partners links that is used to intake different web services into the process. BPEL process is a centralised control point of web service composition.

2. Web Service Composition Modelling Methods based on Cooperation

The service composition method based on cooperation is used to construct composition service model by decrypting the message exchange sequence of component service. The method is similar to the description way of commercial agreement in the e-commerce, and the method believes that it can define their collaborative behaviours by describing the message interaction specification followed by each component of the composition services. This composition approach is focused on describing behaviour feature of message exchange of each component service during the process of service composition process, which is a more direct modelling composition method for the cooperation process participated by many parts.

WSCI is a synthetic language specification of web service based on XML format and jointly developed by SUN, SAP, BEA as well as Intalia [6]. It focuses on tracking message interaction sequence of the messages in cooperation web services, specifying information exchange process participated by the combined web services, and supporting constraints relationship of messages, the message interaction sequence, exception handling, transactional attribute and description of dynamic synthesis.

3. Web Service Composition Modelling Methods based on Planning

The composition method based on planning brings the thoughts of classical AI (Artificial Intelligence) into the technical service composition. For the service composition based o AI planning, the initial state and target state are defined by the requirement of synthesis services, the action is a set of available component services, rules for state transition define the antecedent, and consequent of service function of each component [73]. Therefore, the process of web service composition is to find a group of services from the optional services and make the functions of the group service composition to meet the requirement specifications of the combinational service. The composition method based on planning is required to have the aid of research method of AI, and combined with the semantic web techniques [116], such as OWL-S [24], SWSI (Semantic Web Services Initiative) [28].

The Stanford University has developed a variety of techniques based on artificial intelligence planning, using Golog to automatically combine web services systems [74]. Each web service is considered as an action, an atomic one or a complex one, the complex actions are combined by several atomic actions. A composite service is a series of set of atomic web services connected by programming languages symbols. In this combined system, it still needs to make-up semantics of web services, establish ontology library, knowledge base of intelligent agents and so on.

The composition of web services increased flexibility, reusability and integration of web services. The existing modelling method for web service composition can make the basic web services to effectively compose a new web service which better-meet the requirements and more powerful. But these web service compositions are carried out according to functional fragments to achieve business functions of specific requirements. They are more independent and looser. Web service composition can be deemed as a systematic method for reusing the basic web services to build system rapidly and effectively. So it needs the support of system-level modelling.

3.5 Summary

This chapter introduces and discusses the state of the art of the related fields including MDD, MDA, DSM, web services and web service composition, etc.

- Software development is switched from code-centric to model-centric with MDD. MDD focuses on system modelling based on the best practices, and guides every phases of software development with models. Model is not only an analysis and design specification, but also a software product which can be automatically transformed into the executable system. That is the target of the thesis.

- OMG puts forward MDA system and makes some standards to support MDA. However, the biggest problem of OMG is that it elicits the whole architecture, but does not provide the concrete implementation. But MDA system is also a well reference of the thesis. Especially, ASL and OCL are referred to support action specifications and model constraints in XDML design.

- DSM pays more attention to the small and proficient modelling. The goal of DSM is system implementation rather than system analysis and design. DSM puts forward a three-tiered architecture in the target environment including language, generator, and domain framework, which are the foundation of the thesis work. In the thesis, XDML language is designed for describe the executable domain-specific model; the model parsing and executing mechanism is used by DSMEI to replace the code generator; domain framework is contained in DSMEI to support model execution.

- Web services can simplify the complex software applications, and support resources sharing and cooperating work in the distributed environment. In the thesis, web services are adopted as the core functional implementation entities of xDSM execution supported by DSMEI. xDSM can be transformed into the service-oriented domain-specific application by parsing and executing the behaviour logic of xDSM models in DSMEI.

The thesis integrates DSM and web service techniques with MDD and proposes a unified approach, SODSMI, to build the executable domain-specific model and achieve the target of MDD. The details of the approach will be given in the next sections.

Chapter 4

Proposed Approach

4.1 From Models to System Implementation

4.1.1 Problems

The thought of MDD is pioneering but there are some problems during the process of the traditional MDD. The core problem is that the current models are difficult to be transformed into system implementation. The main reasons are concluded as follows:

1) Current models are too abstract

MDD needs to do software modelling. The modelling activities involve structure modelling and behaviour modelling. The structure modelling is the foundation for supporting software behaviour. It defines the bound of the software behaviours. The behaviours define software functions and realise the system objective. Therefore, the behaviour semantics play a decisive role within the transformation from models to system implementation. Current behaviour modelling itself is highly abstract and omits many behaviour details definition. So the behaviour semantics of most of behaviour model elements are imprecise, uncertain and ambiguous. Behaviour models are only used as a guideline for system implementation.

2) Current modelling domains are too wide

The universal modelling method is the mainstream of current modelling methods.

The development of software systems are becoming more and more complex and the involved domains are larger and larger in range and amount. So the universal modelling, the modelling language has to be modified and increased accordingly. And models are increasingly large, difficult to be used and understood as well as more difficult to be transformed into system implementation.

3) Current modelling activities focus on system analysis and design, not system implementation

There are different model views of software system at every phase in software development life cycle. Most model views focus on system analysis and design so as to be used for developers to communicate with each other and carry out the specification design. There are little model views for system implementation.

4) The supporting environment for model implementation is absent

To realise models depends on the specific supporting environment for model implementation, such as code generator or model virtual machine. However, the supporting environment for model implementation is difficult to achieve due to the localisation of models themselves. More commonly, models are only used to generate parts of software products, for examples, code framework, documents, configuration scripts, etc.

4.1.2 Characteristics of Domain-Specific Modelling

Compared to the large and universal modelling of UML, DSM pays more attention to the small and proficient modelling. The goal of the methodology of DSM is system implementation rather than system analysis and design. The characteristics of DSM are summarised as follows:

1. Lower complexity

DSL is customised for solving software development problems existing in a certain application domain. It is a specific and problem-oriented language [19]. It does not require that the target range of DSL covers all the software problems. Once a DSL is correctly formed, it should involve the terminologies and concepts of the specific problem domain. Namely, it means that a DSL may be useless for other problem domains. Though DSLs give up the universal scope of the language, it improves the description accuracy of the specific domain problem and its solutions, and reduces the complexity of the language itself. DSLs are simpler and more accurate in syntax and semantics than the universal modelling languages. That reduces the difficulty of DSLs compiler, interpreter and the supporting environment development.

2. Higher abstract level

DSL is the core of DSM. It is a language for solving domain-specific problems. It provides suitable and fixed abstract concepts and notations of the domain. It provides the concepts and rules which represent the corresponding application domain rather than the concepts and rules which are in a given programming language. Therefore, DSL is at a higher abstract level.

Generally, the main domain concepts are mapping to the objects of the modelling language, while other concepts are mapping to the attributes, relationships, sub-model of the object or model links of other languages. Therefore, DSL makes developers use domain concepts directly to construct the domain models. It is able to describe domain concepts, the relationships between domain concepts and domain rules with larger granularity. Developers can use the domain knowledge elements in DSL directly to develop the application system, rather than develop program code or components that are corresponded to domain concepts from the most basic classes or objects from the scratch. So the system development efficiency is effectively improved.

3. Integrity of MDD

In DSM based software development, developers just need to use DSL to carry out modelling. After modelling completed, these models can be automatically transformed into the executable code. From the perspective of modellers, it is integrated from modelling to system implementation. DSL is the foundation to generate the integrated code automatically. Models are used for both design and system implementation at higher abstract level. The realisability runs through the entire modelling life cycle. It is the real MDD. The main aspects that DSM differs from the early CASE and UML tools are: code generator is built in house. Namely, it is written by the experts who have development experiences for the same domain, but not provided by vendors. Code generator built by experts can be adjusted to adapting to an application system. It is with strong customisability. The code generated based on DSM is practical, readable, and efficient. It looks like that the code are written by the developer who defines code generator. So the integrality of model itself and that of the model implementation foundation are ensured.

4. Goal for system implementation

The goal of DSM is system implementation rather than system analysis and design. To build the domain-specific meta-model, construct the relevant DSL and build code generator are all customised for the specific domain and aim at how to make models transform into the executable code. During the software development process based on DSM, models are main products. Models specify not only what the system will do, but also how to do. What the developers have is the source model not the source code. Therefore, any modification to the system also is the modification to models, rather than modify the generated code. In the process of adopting DSM, developers / modellers only use the corresponding tools to carry out modelling. Once the functional requirements of a particular system and the logical relationships between them each other are completed, the modelling is finished. Then models can be automatically transformed into the executable code by code generator.

5. Capability of meta-modelling

In the DSM methodology, the modelling process is divided into two steps: the first is to carry out modelling for domain concepts and domain rules that may exist in the target application domain, which builds the domain meta-model. The second is to use the result of meta-modelling (DSML) to carry out domain application modelling for the target application system.

The meta-modelling supporting capability is the main task of DSML. One of the goals of DSM is that end users can take part in the application development and adjust the application logic so that developers can focus on the development for DSML supporting tools rather than struggle for meeting the continually changed application requirements. End users use DSML to take part in the development and maintenance of the application software. The software requirements which are easy to change can be realised in the application modelling and controlled by end users themselves. So developers can focus on the more valuable work. Therefore, the meta-modelling supporting capability of DSML is emphasised in the DSM methodology, which makes end users get the greater flexibility in the modelling language to adapt to the different domain application requirements.

6. Reusability

A specific domain is not a specific industry domain, but a functional domain covered by a group of application systems which have the similar functional requirements [64]. Software reuse for a specific domain is relatively easy to achieve [132]. The cohesion (the compact correlation of domain knowledge on logic) and stability (in a certain period, domain knowledge do not change acutely) of a specific domain provide software reuse activities with the reusable software assets so as to make domain-specific software reuse relatively easy to achieve. Domain engineering is the main technical means to generate the reusable software assets, which includes the three phases: domain analysis, domain design and domain implementation. Actually, domain meta-models and DSML generated by DSM are the expressions of domain knowledge. These domain knowledge and the reusable software assets that are used to realise the domain knowledge will be ceaselessly reused in the different application modelling processes.

4.2 Proposed Approach

4.2.1 Targets and Ideas

The role of model for software analysis and design is irreplaceable. Developers establish software analysis and design models in accordance with a variety of software standards, and communicate with each other by models. Model is expected to bring an essential leap of software development, and drive the whole software development process. It means that modelling is not only related to the requirement analysis, software design and software implementation, but also able to support unit testing, system testing, long-term system maintenance and software reuse, etc. The above all require the executability of model. Only executable models can strictly ensure that model validation, system-generation and system maintenance are based on the models.

The key elements of the executability of model lies in whether there are a well-defined models and whether there is a code generator which can automatically and completely generate code. Both of them are mutually constraining and complementary. Code generator can be simple and easy to implement while the model is complete and accurate. On the contrary, code generator must be difficult to achieve with complex structure and required adaptability and flexibility while the model is imprecise. In order to build the executable model, and achieve the automatic transformation from models to system implementation, there are two aspects both need to be concerned. On one hand, models ought to be refined and the degree of abstract ought to be reduced so that models can gradually approach system implementation; on the other hand, code generator ought to have strong adaptability and flexibility to reflect the model description.

The thesis is based on domain-specific modelling: the executable model which is in accordance with MMLs 5 is established with behaviour modelling as its core. Based on the complete, consistent, detailed and accurate model description by XDML, model parsing and executing mechanism are used to replace code generator, and combine with Domain Framework as the infrastructure of the domain-specific model implementation. Different from other domain specific modelling approach, the abstract level of code implementation is enhanced by the standardised, self-contained, self-describing, modular web services. Encapsulating the details of code implementation, the related domain-specific software functional entities are provided to DSMEI (Domain-Specific Model Execution Infrastructure) by the way of web services cluster. The system running is driven by parsing and executing the behaviour models. The above is the core idea of the thesis. The framework of SODSMI (Service Oriented executable Domain-Specific Modelling and Implementation) is shown in Figure 4.1.

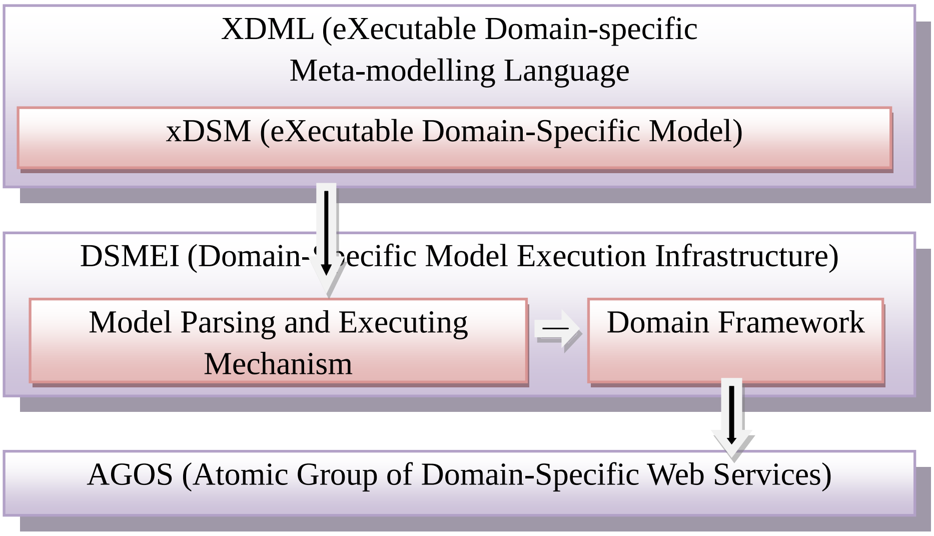

XDML (eXecutable Domain-specific Meta-modelling Language
xDSM (eXecutable Domain-Specific Model)
DSMEI (Domain-Specific Model Execution Infrastructure)
Model Parsing and Executing Mechanism
Domain Framework
AGOS (Atomic Group of Domain-Specific Web Services)

Figure 4. 1 Framework of SODSMI

SODSMI constructs executable models and their execution infrastructure based on domain-specific modelling through the model refinement and the enhancement of code implement.

From the perspective of functionalities, SODSMI is divided into three levels, corresponding to four core elements:

- xDSM -- Executable Domain-Specific Model
- XDML -- Executable Domain-specific Meta-modelling Language
- DSMEI -- Domain-Specific Model Execution Infrastructure
- AGOS -- Atomic Group of dOmain-specific web Services

XDML is used to describe xDSM. xDSM is parsed and executed in DSMEI. Its execution depends on the corresponding interfaces provided by Domain Framework. Domain Framework provides the core software functional entities through domain-related services of AGOS, and supports the xDSM execution upwards. xDSM, XDML, DSMEI and AGOS constitute the framework of SODSMI together.

4.2.2 Features of the Proposed Approach

The SODSMI framework is aimed at modelling for system implementation, which reduces the model complexity and improves the model accuracy. This method has a holistic and sustainable system to support the transformation from models to system implementation. Compared to other modelling methods, such as MDA system, the proposed approach is more suitable for the establishment and support of executable models, mainly shown as follows:

(1) SODSMI is customised for solving software development problems in a certain application areas. It is dedicated and problem-oriented. Although it is at the expense of commonality, it improves the accuracy of the description on domain specific problems and its solutions, and reduces the complexity of modelling.

(2) SODSMI improves the abstract level of models, and XDML provides an abstract mechanism to deal with the complexity of specific domains. It provides concepts and rules of the corresponding application domain, rather than those of a certain given programming language. Modellers face the domain concepts with different granularity directly, rather than construct the implementation details in the light of classes and objects, etc.

(3) SODSMI pays attention to the integrity of MDD. Its goal is to achieve the system implementation, rather than to simply use models as a means of analysis and design. SODSMI completes the whole process from model establishment to code generation.

(4) SODSMI emphasises on the capacity of meta-modelling, and adopts the separation of meta-modelling and domain application modelling to establish models that adapts better to specific domain. At the same time, it is able to separate users’ application modelling from domain experts' meta-modelling as well as developers’ creating support tools.

(5) In SODSMI, the establishment of meta-model and code generator are developed within the organisation. They are mutually complementary: the model establishment is adapted completely to code generator; the generated code is practical, readable, and efficient as same as the code is written by experts who define the code generator. Meanwhile, the establishment of meta-model and code generators implicates a lot of implicit implementation convention that need not be expressed at the model layer, which observably reduces the complexity of models.

(6) SODSMI is based on domain engineering, which provides a well support in essence for software reuse; on the contrary, the software reuse techniques also provides a well support for the DSM method.

4.2.3 Executable Domain-Specific Model

The primary task of SODSMI is to build executable models, while the executability of model is always an underbelly of MDD for a long time. Software itself is dynamic. Static models can describe some profiles of software, for examples, the subordinate structure and the system hierarchy. But it can describe neither the entire software, nor the running process of software. At the same time, the abstract of models restricts the accuracy of models, which makes models lack of many of the key elements that are used to construct entire software. In MDA system, UML can be used to build models of the system from different perspectives and aspects. Model views represent a part or a profile of the system. However, there are neither positive connections nor constraints among those model views. Model views can be more or less, be concrete or abstract. The process of building a model can be ceased at any phase. It is very difficult for modellers to construct a complete software model unless they understand all the details of code generator. That makes the executable models difficult to achieve in UML system.

xDSM is constructed based on the domain-specific model, and is technically applied to solve the software development problems existing in a certain application domain. xDSM represents the concepts and rules of the domain. The model is targeted, that narrows the scope of the description effectively and is helpful to define the model accurately. xDSM modelling process is divided into two phases: the xDSM meta-modelling phase and the xDSM application modelling phase. The former is carried out by domain experts and technical experts, and the latter is carried out by end users. The duty and the role of modellers in each modelling phase are different, as shown in Figure 4.2:

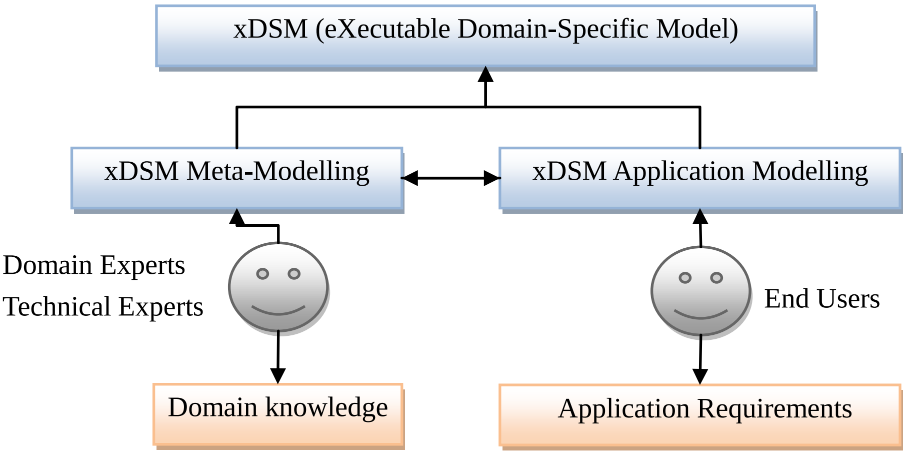


Figure 4. 2 xDSM Meta-Modelling and Application Modelling

xDSM is required to meet MMLs 5 standards. It requires the model definition is sufficiently precise. The accuracy here is to describe the details relevant to the modelling objectives accurately rather than to describe all aspects of modelling. The core of xDSM is behaviour modelling. It is required to describe domain concepts and system behaviours unambiguously. In the meta-modelling phase, domain concepts are described unambiguously, including domain objects, relationships, constraints and any operations embodied in the domain concept. In the application modelling phase, the target is to meet all the requirements to software systems. The accurate software behaviour modelling is carried out by using meta-model. The model does not care about the implementation of local software functions, but it does not ignore the necessary details of the behaviour execution yet -- the data flow, the control flow and the related constraints of behaviours must be described in detail.

On one hand, the measurement of the accuracy of models is determined by domain experts and technical experts through xDSM meta-modelling and DSMEI. Namely, if the application model which is built according to the definition of the meta-model can be accurately and completely executed by DSMEI, the models can be regarded accurate enough. On the other hand, the application model which is built in accordance with end users' requirements can ensure the integrity of the model. Namely, if the results of the application model execution meet the system requirements completely, or the generation system realises the functional requirements completely, the models can be regarded complete enough. Moreover, application modelling also facilitates the improvement of meta-modelling and the execution environment, to meet the requirements to application modelling better.

Furthermore, the description of the behaviour details in xDSM also increases the complexity of modelling. It requires to adjust the complexity of modelling through meta-modelling and application modelling. That is guided by domain experts and developers mainly in the meta-modelling phase. On one hand, the behaviour complexity is encapsulated in the meta-model while the behaviour details are hidden in domain objects and relationships with the different granularity; on the other hand, the complex behaviour descriptions are hidden by the implementation convention of the meta-model and the execution environment. So end users can do the application modelling simply and flexibly. So it is easier for end users to build the executable model with high-quality.

4.2.4 Executable Domain-specific Meta-modelling Language

Following the guide of MMLs5, XDML is defined to describe xDSM meta-model and its application model accurately. XDML extends the semantic basis of XMML language -- a visual meta-modelling language, and integrates the well-defined behaviour semantics to support the domain-specific behaviour modelling. XDML defines the concrete syntax of AS&MC which provides accurate definition for dynamic behaviours of models.

XDML improves the description accuracy of the specific domain problem and its solutions, and reduces the complexity of the language itself. XDML is simpler and more accurate in syntax and semantics than the universal modelling languages. That reduces the difficulty of XDML compiler, interpreter and the supporting environment development.

XDML is at a higher abstract level. Generally, the main domain concepts are mapping to the objects in XDML, while other concepts are mapping to the attributes, relationships, sub-model of the object or model links of other languages. Therefore, XDML makes developers use domain concepts directly to construct the domain models. It is able to describe domain concepts, the relationships between domain concepts and domain rules with larger granularity morpheme. Developers can use the domain knowledge elements in XDML directly to develop the application system, rather than develop program code or components that are corresponded to domain concepts from the most basic classes or objects from the scratch. So the system development efficiency is improved effectively.

4.2.5 Domain-Specific Model Execution Infrastructure

Today, the scales of software systems are increasing, and the number of people who are involved in software applications is also increasing, so as to make software architecture more and more complex. The software is no longer limited to a stand-alone desktop system, but gradually evolved into the networked and complex systems which are integrated with each other. In this case, the functionalities of code generator are limited because the generated code may be only a part of the complex software system. Moreover, code generator is also a software product. It is more complex than the generated system, and it is also needed to face the changes of the generated system itself, that requires code generator to be strongly adaptable and flexible.

DSMEI is combined with Domain Framework, and employs the model parsing and executing mechanism substituting the code generator to execute xDSM models directly. Domain Framework is used to provide the interface of the underlying platform to the generated code. DSMEI encapsulates the architectures, platforms and concrete implementation of the domain-specific application system into Domain Framework, which reduces the complexity of the generated code significantly, as shown in Figure 4.3.

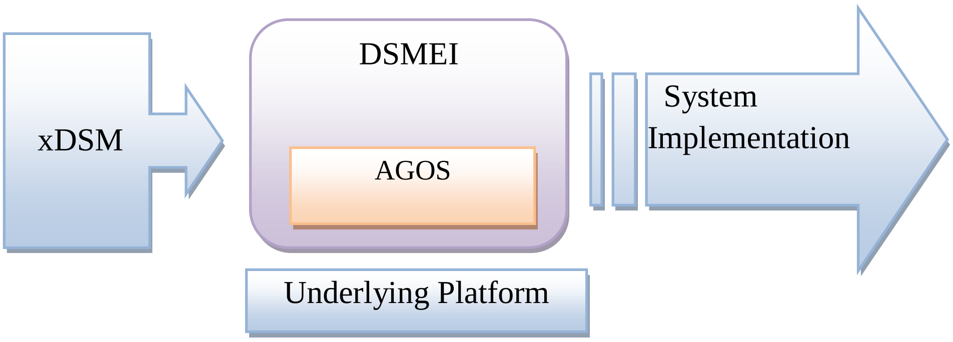


Figure 4. 3 DSMEI Functional Structure

The system behaviours are able to be described by xDSM completely and accurately. Based on that, the model parsing and executing mechanism is used by DSMEI to replace the code generation process. xDSM is parsed into the operations with precise semantic, and the operations are corresponded to the interfaces provided by Domain Framework. Here the model itself is an executable software product. As the evolution of Domain Framework is independent of the parsing and executing of the model, the model can be transform into the system implementation on DSMEI dynamically and flexibly. Furthermore, DSMEI is combined with Domain Framework, and encapsulates the parts of domain-related implementation into the modular web services through AGOS. So that it can focus more on the parsing and executing of the model, as well as the combination with web services which are related to the specific domain. That makes the architecture of DSMEI general, while the dynamic characteristics and the virtualisation techniques of web services make DSMEI more flexible, so that a common and flexible supporting environment is provided for the model execution by this way.

4.2.6 Software Function Entities - Web Services

To a certain extent, the code is also a model. It is the most refined model, and a language description defined precisely. It can be used to describe a system, but it is also platform-dependent. But such an iterative refinement is not necessary. On one hand, over-refinement makes the scale of model so large that the model loses its abstract nature. On the other hand, to deal with the ever-changing system requirements, even if the advanced language also needs to be added SDK (Software Development Kit) continuously, it must be much harder to the model which only have a weaker descriptive ability. Consequently, a better software functional entity must be found to realise the executable model.

The software functional entity has undergone several evolutions: from functions to objects, from objects to components, then from components to web services. Web services architecture adds and standardises a new layer, named "Service Layer" between the logistic layer and technical implement layer. The standardisation and dynamic characteristics make web services be able to provide the abundant and flexible software functional entities. AGOS adopts web services that is standardised, self-contained, self-described and modulised to enhance the abstract level of the code implementation, encapsulates the details of the code implementation, and provides the related domain-specific software functional entities to DSMEI by the way of web services cluster. Web services are not stand-alone. They depend on the domain-specific application systems and their processes. The development and reuse of web services have already been determined when the xDSM meta-model is constructed. It is a top-down design process. Based on the domain concepts, it describes the domain behaviour process dynamically according to the model, and drives the definition and functionalities of web services according to the realisation requirements of the model. The design principles of web services are as follows: the common parts of the specific domain are encapsulated into web services. The changeable parts are divided into two kinds: one kind that is easy to deal with by xDSM is defined directly by model; the other kind that it is not easy to deal with by xDSM will be transformed into service parameters, and use the parameterised means to handle the change-point. Web services provide the minimal software functional entities in the entire system. It is also the implementation foundation of the entire executable model.

Various web services at the different levels are required to support the problem space involved in the domain-specific modelling. AGOS regards a group related web services of a specific domain as a service cluster. On one hand, it requires a lot of web services entities to provide different functions; on the other hand, there may be several corresponding web services entities to the same functional requirement. So DSMEI is able to not only support the protocol of the service itself, but also deploy web services cluster dynamically in the software life cycle, for examples, querying services, matching services, assembling services, replacement services, load balancing of the service group of the same functional node, and adjustment of the coordinated services, etc. The flexible architecture of DSMEI is the foundation of the above all. It is able to provide Domain Framework dynamically based on web services, and adjusts the existing web service cluster to adapt software changes quickly.

4.3 Summary

In this chapter, after debating the problems emerging in the process from models to system implementation, and analysing the characteristics of DSM, the framework of SODSMI is proposed. The SODSMI framework is aimed at modelling for system implementation, which reduces the model complexity and improves the model accuracy.

From the perspective of functionalities, the SODSMI framework involves four core elements: xDSM, XDML, DSMEI and AGOS. They will be described in detail in the thesis.

- xDSM modelling process is divided into xDSM meta-modelling phase and the xDSM application modelling phase. The duty and the role of modellers in each modelling phase are different. Behaviour modelling is the core of xDSM modelling.

- XDML improves the description accuracy of the specific domain problem and its solutions, and reduces the complexity of the language itself. XDML is simpler and more accurate in syntax and semantics than the universal modelling languages. That reduces the development difficulty of XDML compiler, interpreter and the supporting environment.

- DSMEI employs the model parsing and executing mechanism replacing the code generator to execute xDSM models directly. DSMEI is combined with domain framework introduced in DSM. It encapsulates the architectures, platforms and concrete implementation of the domain-specific application system into domain framework, which reduces the complexity of the generated code.

- AGOS adopts web services to enhance the abstract level of code implementation, encapsulates the details of code implementation, and provides the related domain-specific software functional entities to DSMEI by the way of web services cluster. Various web services at different levels are required to support the problem space involved in the domain-specific modelling. AGOS regards a group related web services of a specific domain as a service cluster.

XDML is used to describe xDSM. xDSM is parsed and executed in DSMEI. Its execution depends on the corresponding interfaces provided by domain framework. DSMEI provides the core software functional entities through domain-related services of AGOS, and supports the xDSM execution upwards. xDSM, XDML, DSMEI and AGOS constitute the framework of SODSMI together.

Chapter 5

eXecutable Domain-Specific Model

5.1 Keys to xDSM

MDD is a model-centric method for driving the whole process of software development. It is difficult that a model is transformed into the concrete realisation if the model is ambiguity and at a high abstract level. Through analysing the core value of model and Modelling Maturity Levels, the thesis proposes the SODSMI framework, in which executable models are established in the means of DSM, and the system is realised with the support of DSMEI. The keys to making xDSM models executable are the accuracy and integrality of model, and behaviour modelling. They all are built based on domain-specific meta-modelling.

5.1.1 xDSM Meta-Modelling

In the domain-specific software development, it is required to define the special modelling language and establish the corresponding modelling environment for the different application domain. But it costs much higher to develop special modelling tools for different modelling languages. The meta-modelling technique is a good solution to this problem [122]. The main idea is that the domain-specific meta-model is customised by domain experts according to the requirements of specific domain, and the meta-model is parsed by the corresponding tools. So a DSML language needs to be elicited to support the meta-modelling, and the modelling tools need to be developed to support the DSML language. A large number of engineering practices show that the efficiency of domain-specific modelling based on meta-modelling is 10 times higher than that based on UML [77]. There are two kinds of meta-modelling frameworks [66]:

- ➢ Modelling method with generic modelling as the core: a meta-model which is used to describe a modelling language is established by domain experts with generic modelling tools. It is configured for the generic modelling tools to make the generic modelling tools support the modelling language described by the meta-model. The generic modelling tools are also known as GME (Generic Modelling Environment). GME can be used not only to create meta-models (meta-meta-models are configured for the generic modelling tools), but also to build application models (meta-models are configured for the generic modelling tools) [62].
- ➢ Meta-modelling based on the modelling tool generator: The first step is to establish meta-model by the modelling tool generator to describe the modelling language. It does not produce the configuration files for the modelling tools during this process, but generate the modelling tools directly which support that modelling language.

In this thesis, the modelling method with generic modelling as the core is used to build the executable model, and define xDSM meta-model and xDSM application model in a unified generic modelling environment. A fixed generic modelling environment can be integrated well with DSMEI, which is convenient for model validation and testing. At the same time, GME can make the integration of xDSM meta-model which is corresponded to domain spaces be realised.

Domain-specific meta-modelling is an approach of the systematic model abstract. The abstract is able to reduce the complexity of models and modelling language while it is used to describe system characteristics and maintain the validity of the model. The xDSM modelling process is based on the domain-specific meta-modelling approach, which is divided into meta-modelling phase and application modelling phase while the roles of modellers separated at the same time, as shown in Figure 5.1.

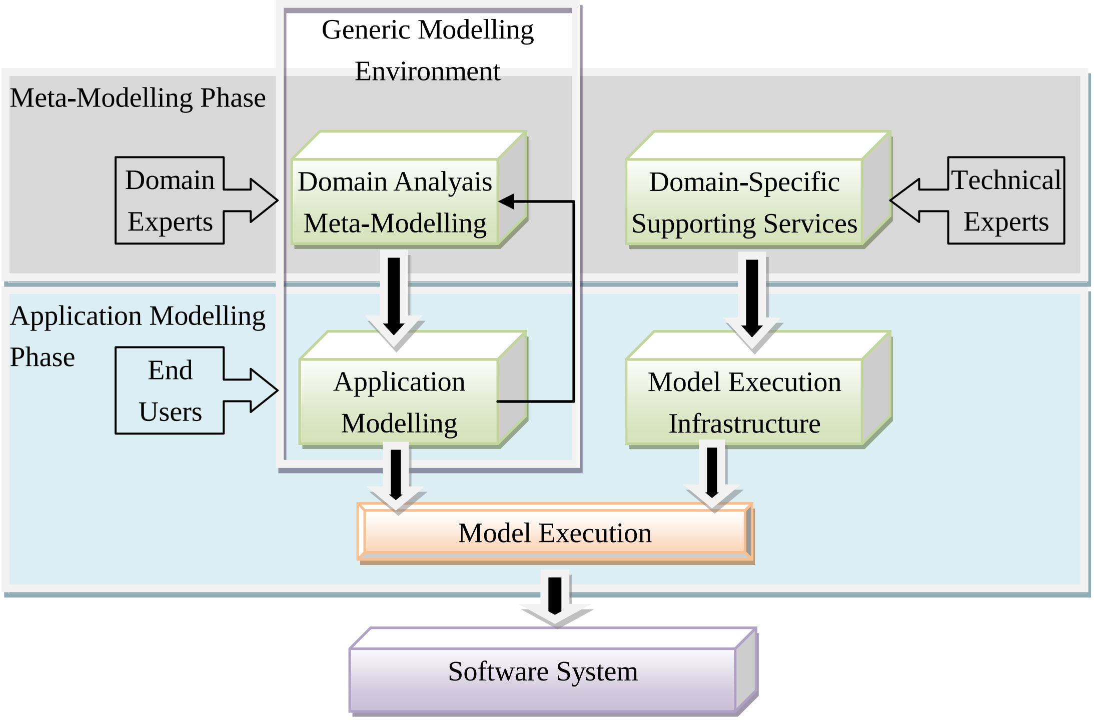

Generic Modelling Environment
Meta-Modelling Phase
Domain Experts
Domain Analyais Meta-Modelling
Domain-Specific Supporting Services
Technical Experts
Application Modelling Phase
End Users
Application Modelling
Model Execution Infrastructure
Model Execution
Software System

Figure 5. 1 xDSM Meta-Modelling Process

In the phase of xDSM meta-modelling, domain experts analyse the specific domain and establish xDSM meta-model. In other words, domain experts construct models of domain knowledge, extract domain-specific concepts, constraints, rules and the form of representation, and create domain objects, relationships and the related constraints. According to xDSM meta-model, domain-specific supporting services are developed by technical experts at the same time. Meta-modelling and the development of domain-specific supporting services are complementary. While meta-model built in the top-down way determines the requirement specifications and organisational relationship of domain-specific supporting service, the execution of xDSM application model which is built based on xDSM meta-model is also supported by domain-specific supporting services. Moreover, xDSM meta-modelling and the development of domain-specific supporting services are negotiated and completed by domain experts and technical experts together. During the process, it is involved with many implicit conventions and constraints to ensure that xDSM application model built according to xDSM meta-model can be executed normally with the support of domain-specific supporting services.

In the phase of xDSM application modelling, corresponding to application requirements and based on xDSM meta-model, end users use domain-specific concepts to carry out the application entity modelling for the problem domain. The specifications and constraints defined by xDSM meta-model must be abided strictly in the modelling process. xDSM application model established by end users can be executed in DSMEI so as to validate users' application requirements, and ensure that application modelling can meet the requirements of software system completely.

Through the separation of meta-modelling and application modelling as well as the role division of modellers, the responsibility of each role can be defined. By integrating system modelling and modellers for xDSM modelling, the maximum value of each role can be brought into play in MDD. The domain knowledge is modelled by domain experts, and the software is controlled and adjusted by end users according to software requirements. So that technical experts and developers can concentrate on the development of DSMEI and domain-specific supporting services. The more are controlled by xDSM, the cost of software development and maintenance will be lower, thereby the software productivity is maximised.

5.1.2 xDSM Behaviour Modelling

Software is dynamic and composed of various behaviour sets which accomplish the different system objectives. Software specification is objective-oriented because only system objectives are the most direct expression of software system [131]. A system objective is achieved by a number of domain main concepts working together.

Behaviour is the main expression of the system objective. A series of actions are executed in software specification to achieve the system objective. To extract the behaviour model corresponding to system objectives and to describe system objectives with behaviours are the keys to the problem-oriented modelling method.

The software behaviour is divided into two basic types, the state-related behaviour and the state-free behaviour. The state-related behaviour can be expressed by finite-state machine, and the state-free behaviour can be expressed with operations. The most of software behaviours are state-free. To the state-related behaviour, it is understood here as follows: being given a message, the responding behaviour of the state subject is decided by its current state. The state-related behaviour can be also expressed with operations which is the outcome from parameterising the states and merging the state transition operations. Consequently, software behaviours can be expressed with operations entirely.

The behaviour structure is composed of behaviours, actions and operations, which are the keys to the domain-specific behaviour modelling, as shown in Figure 5.2.

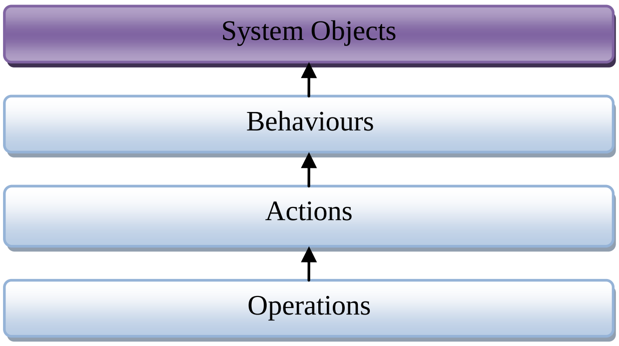


Figure 5. 2 Behaviour Structure

Behaviour is the direct result of actions of at least a domain concept. Behaviour does not exist by itself. It must depend on domain concepts and actions. Action is the basic unit of behaviour, which is contained in behaviour. Behaviour provides execution context and constraint for action, and decides their coordination and the execution timing. Action is more concerned about the independence and atomicity of semantics which is built based on the conceptions that are proved in computer science. Operation is the main representation of action and the basic unit of action specification. An operation gets a group of inputs, which is transformed into a group of output by executing actions. All the input and output can be defined and described by the value specification in detail. The operation is similar to the concept of procedure, operation, or subroutine in a programming language at many aspects. It has the following features:

- Operation is executed synchronously and asynchronously according to requirements;
- Operation can have zero or several input parameters;
- Operation has one output at most, or exception;
- The input and output parameters can be any valid data type.

The behaviour modelling of xDSM is carried out according to behaviour logic, not the simple expression logic and computational logic. Behaviour and computation are blended with each other. For decoupling the behaviour logic and the computational logic, the logical behaviour can focus on describing the coordination relationship between the domain concepts, while the computational process of the implementation details can be ignored. And the computational logic of the atomic operation of domain business can be encapsulated in the services. So the behaviour logic based on the above can be modeled, configured, and dynamically loaded.

- Can be modeled: after encapsulating the atomic operation of domain business as services, the behaviour description is carried out according to the coordination logic of domain concepts. That simplifies the complexity of the behaviour description greatly, and makes the modelling possible.

- Can be configured: through decoupling the logic behaviour and the computational logic, the atomic operation services of domain business can be configured. And the behaviour model can be adjusted through the configuration.
- Can be dynamically loaded: the behaviour model can be corresponding to the different atomic operation services of domain business. They can be substituted at run-time since the flexible connection between the behaviour model and the services, so that the dynamic loading can be realised.

Based on the decoupling between the behaviour logic and the computational logic, Behaviour Scenario is introduced in the thesis for behaviour modelling of xDSM, as shown in Figure 5.3.

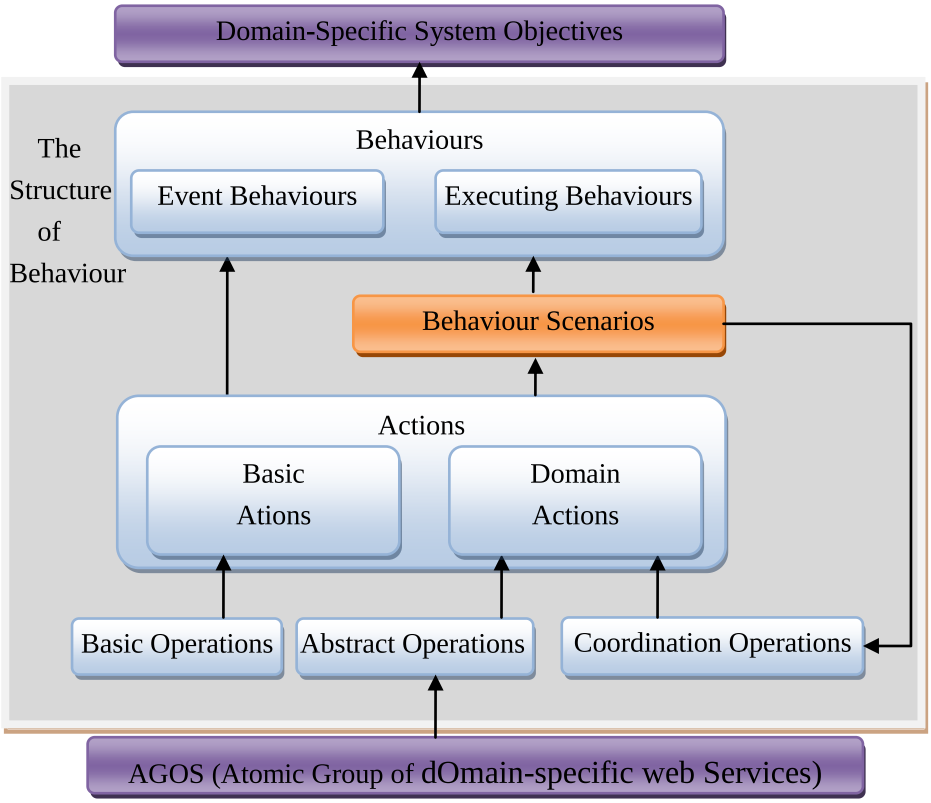


Figure 5. 3 The Behaviour Structure of xDSM

Behaviour, action and operation are the main bodies of the behaviour structure of xDSM. Behaviour Scenario is used as the view of behaviour modelling. It focuses on:

- Constructing behaviour models according to the domain-specific system objectives, and describing system objectives with behaviours, thereby, the software system can be described.

- Behaviour modelling of xDSM can be divided into two types: Event Behaviour and Executing Behaviour. Executing Behaviour describes the set of Executions of domain concepts. Execution is to realise an executing process according to a definite strategy. Executing Behaviour is the behaviour executed by domain object itself or the cooperative behaviour between the domain concepts. Event Behaviour describes the set of Occurrence of domain concepts. Occurrence is produced within the system as well as can affect the system. Event Behaviour and Executing Behaviour are described with Operation. Both they are related to the concrete value specifications, execution specifications and domain concepts, and can be modeled in Behaviour Scenario.

- Action is the basic behaviour unit of xDSM and the basis of behaviour semantics. Action can be used to construct the behaviour directly which complete a certain business objective. A complex behaviour can be also completed by several Actions working together. Actions are divided into Basic Action and Domain Action. Basic Action executes the basic action of the supporting behaviour which is provided by the execution framework itself, such as Exception Action, Variable Action, and Message Action, etc. Domain Action is formed according to the domain-specific business objective. It contains the domain objects and the behaviour concepts within relationships. It is the representation of domain-specific behaviour, and composed of Abstract Operation and Coordination Operation.

- Basic Operations support Basic Actions while Abstract Operations and Coordination Operations support Domain Actions collectively. Abstract Operation is an abstract of the concrete implementation operation, which describes the structured interface information of an operation. It is corresponding to the concrete implementation of atomic operation services of domain business. Coordination Operation is constructed based on Action in Behaviour Scenario way, in which it may contain Basic Operations, Abstract Operations and other Coordination Operations. At the same time, Coordination Operation also has the accordant structured interface information as same as Abstract Operation.

- Behaviour Scenario is a container of Actions. It is used to illustrate a series of actions of the system behaviour, and describe an executing system process. It is modelling from the perspective of the domain behaviour process, and provides the execution context for Actions. The entire Behaviour Scenario manifests in the form of Operation which is able to constitute the action directly according to system objectives, and also be transformed as Coordination Operations. It is the main body for describing Coordination Operation.

5.1.3 Accuracy and Integrality of xDSM

The executable model that is in conformity with MMLs 5 is built based on the accurate and integrate xDSM behaviour modelling. MMLs 5 requires that the model can describe the system completely, consistently, detailedly and accurately, and can be transformed into a software system completely and automatically, so as to realise the model execution in real sense. xDSM modelling is targeted well enough to narrow down the description scope of the mode. The most important thing is that xDSM modelling process is divided into the meta-modelling phase with domain experts and technical experts as the core, and the application modelling phase with end users as the core. Through the separation of meta-modelling and application modelling as well as the role division of modellers, the responsibility of each role can be defined, the maximum value of each role can be brought into play in MDD, and the accuracy and integrity of xDSM can be ensured.

- Integrality of xDSM: Corresponding to application requirements and based on xDSM meta-model, end users use domain-specific concepts to carry out the application entity modelling for solving the application problems. The specifications and constraints defined by xDSM meta-model must be abided strictly in the modelling process. xDSM application model established by end users can be executed in DSMEI so as to validate users' application requirements, and ensure that application modelling can meet the requirements of software system completely. At the same time, in the process of application modelling, if xDSM application model correctly constructed by end users is insufficient to achieve the domain-specific system objective, xDSM meta-model and domain-specific supporting services will continue to be improved by domain experts and technical experts. It is an iterative process. It will enhance the overall integrity of xDSM.

- Accuracy of xDSM: In the process of xDSM modelling, to add the definition of action specifications besides the definition of model elements that improves the accuracy of xDSM substantially. In the phase of xDSM meta-modelling, the construction of xDSM meta-model and the development of domain-specific supporting services are negotiated and completed by domain experts and technical experts together. During the process, it is involved with many implicit conventions and constraints. To measure models accurately is determined by domain experts and technical experts with xDSM meta-model and DSMEI. Namely, if the application model which is built on the meta-model definition can be executed by DSMEI accurately and completely, the models will be regarded as accurate enough.

The integrality of xDSM is a subjective and dynamic concept. It requires that end users, domain experts and technical experts work together to construct the complete xDSM which can achieve the domain-specific system objectives. It also requires the overall integrity from xDSM meta-model, xDSM application model to domain-specific supporting services. xDSM meta-modelling is the basis of the accuracy of xDSM. It integrates the collaborative process of xDSM meta-model and DSMEI. Both of them are complemented and collaborate with each other to realise system objectives, reduce the model complexity, and construct the executable model with sufficient accuracy.

xDSM meta-modelling and xDSM application modelling are the main activities for constructing the executable domain-specific model. While the behaviour model is constructed accurately, model constraints and action specifications are also required to define the xDSM meta-model and the xDSM application model accurately. as shown in Figure 5.4.

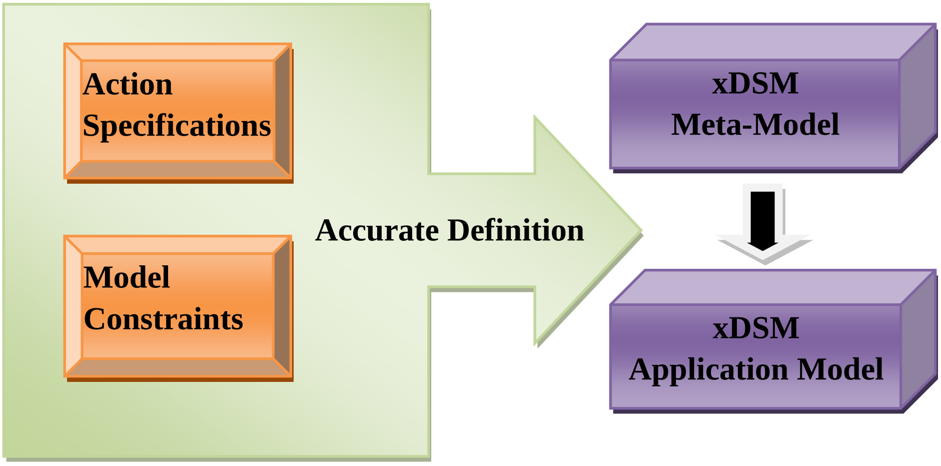


Figure 5. 4 Model Constraints and Action Specifications

In GME, xDSM meta-model is established by meta-modelling language, and xDSM application model is established based on xDSM meta-model corresponding to application requirements. xDSM meta-model and xDSM application model are the main bodies of the executable domain-specific model. Based on behaviour modelling, action specifications provide the unambiguous, accurate and legible definition of the action sequences for the behaviour processing details. It expresses the action details in a clear and accurate way. At the same time, model constraints provide the accurate constraints (semantics conditions or restrictions) in the modelling process to improve the description ability of xDSM behaviour modelling. Action specifications and model constraints can complement the description ability well for the detail parts of behaviour modelling, and improve the accuracy of xDSM significantly.

Behaviour modelling is the core of xDSM modelling. The accuracy of xDSM is to express the behaviour at the necessary accurate level (namely, the requirements of DSMEI to the executable model), not to implement the behaviour. It is required that xDSM can express data flow and control flow accurately. Namely, the data flow and the control flow can be described accurately in the main body of behaviour modelling -- Behaviour Scenario, so that xDSM can be executed correctly in DSMEI. Behaviour Scenario is designed based on the idea of parametric programming. A reusable behaviour scenario can be expressed as a parameterised component. Its behaviour is determined by its parameter values. By parameterisation, the duplication of modelling can be avoided effectively, and Behaviour Scenario can be virtualised so as to make it focus on the behaviour design. Behaviour Scenario represents an independent control flow unit. It is a sequential system within Behaviour Scenario while it is a concurrent system between Behaviour Scenarios. From the perspective of data transferring and processing, Behaviour Scenario relies on the behaviour context and follows the behaviour logic to transform a group of input into an output, thus to achieve system objectives or the specific functional requirements. xDSM carries out the behaviours modelling with Behaviour Scenario, and characterises the data flow and the control flow of behaviour accurately, so as to be executed in DSMEI correctly and completely to meet the requirements of MMLs 5.

5.2 Behaviour Scenario

5.2.1 Behaviour Scenario

Software itself is composed of behaviour sets to achieve the different system objectives. Software specifications are objective-oriented. The system objective is the most direct expression of the software system. Behaviour Scenario involves a series of actions which implement system behaviours. It is used to illustrate interactions and collaborations among domain objects for an executing process of the specific system objective at implementation time. To achieve a business objective, it is possible that several Behaviour Scenarios at different levels are needed to support each other. Behaviour Scenario is used to construct the behaviour model from the perspective of the domain behaviour process. It is a diagram of the behaviour logic. It describes the behaviour logic by the way of visual modelling, defines and restricts the control flow and the data flow of the behaviours accurately by AS&MC syntax. The elements of Behaviour Scenario are shown as Figure 5.5.

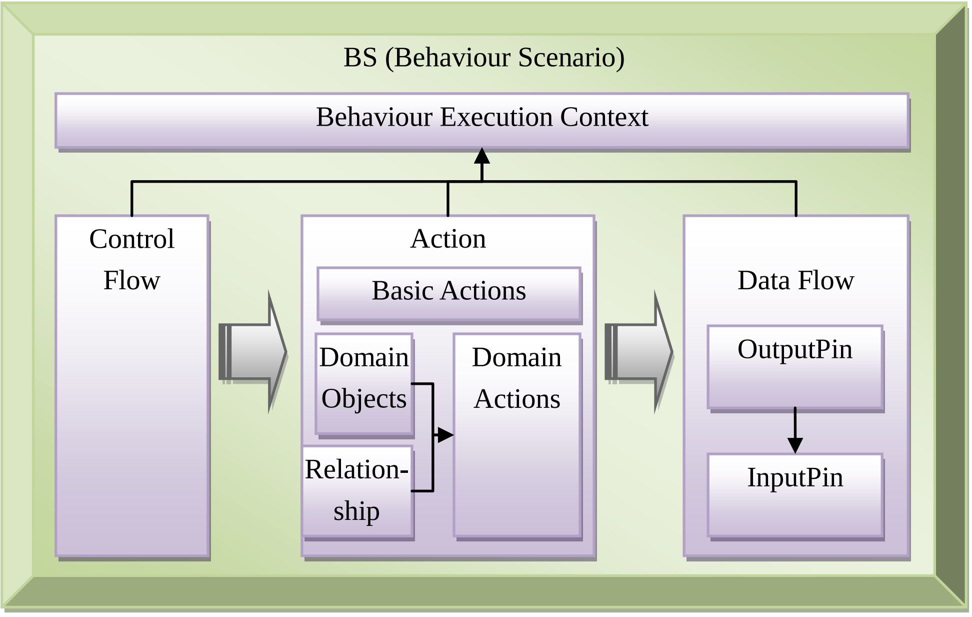


Figure 5. 5 Elements of Behaviour Scenario

Behaviour Scenario executes actions sequentially according to the behaviour logic which is made by the control flow in behaviour execution context. At the same time, it connects the correct data flow by the context as well as the inputting and outputting data of actions to achieve the specific system objectives. The elements involves are:

1. Behaviour Execution Context

Behaviour Scenario is a container of action sets. Behaviour execution context contained by Behaviour Scenario provides the execution environment for Actions. At the same time, it also provides the running environment for the control flow and the data flow of the behaviour. Behaviour execution context of each BS is different at the different execution phases or in the different applications. It reserves the current behaviour states of Behaviour Scenarios. It is the carrier of behaviour logic that implements for the different instances. It involves the domain object instances within Behaviour Scenario, attributes and states of the domain object instances, data items used by Behaviour Scenario, parameter initialisation of Behaviour Scenario, and the inputting and outputting data of actions.

2. Control Flow

The control flow of Behaviour Scenario embodies the behaviour logic. The sequences of actions depending on each other are defined by the control flow, and it has a clear order: the subsequent action is executed only after the previous action is executed. The control flow is an abstract representation of all the possible execution sequences of the action execution. The executing path is controlled by behaviour execution context, runtime constraints and messages. Besides domain entity and relationship, the control element is also the modelling element of Behaviour Scenario, such as the element of condition and the element of loop. The relationship between domain entities extends its constraints and behaviours based on the sequence relationship. Message sending or message receiving which is the description of an occurrence of the system will trigger another control flow or interrupt the current control flow. The execution of operations depends on behaviour execution context.

3. Action

Behaviour Scenario is constructed with actions. Except basic actions, domain actions are attached to entities and relationships or happened between their collaborations. Action is defined within entity or relationship in the form of operation. Entities are the main carrier of actions, which includes active operations and passive operations. Relationships only involve active operations, which are used to describe behaviour relationships between entities. A complex action can be expressed flexibly by action specifications defined by AS&MC syntax can.

4. Data Flow

The data flow reflects the sequential data interaction between actions. The data flow transports data between the actions which are executed sequentially. It has unclear sequence and depends on behaviour execution context. The data flow has its source and target. The source is from behaviour execution context or the output pin of the action, and the target is the Input Pin of the action. Data interactions are formed by the data flow to support the normal turning of the control flow. In Behaviour Scenario, action specifications defined by AS&MC syntax can describe the data flow accurately. The executable behaviour scenario can be constructed only by the accurate data flow definition.

Behaviour modelling for system objectives is carried out by Behaviour Scenarios to describe the behaviour logic. Behaviour Scenario works as Figure 5.6.

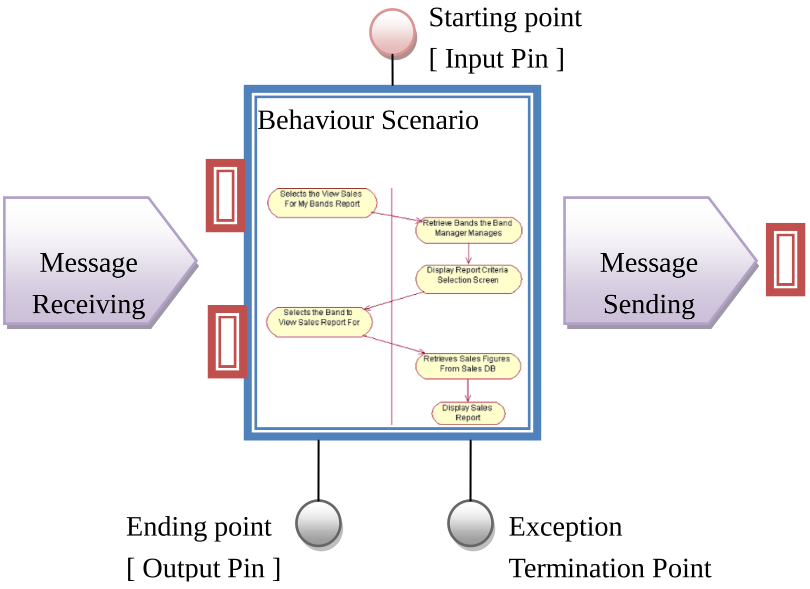

Starting point
[Input Pin]
Behaviour Scenario
Selects the View Sales For My Bands Report
Retrieve Bands the Band Manager Manages
Display Report Criteria Selection Screen
Selects the Band to View Sales Report For
Retrieves Sales Figures From Sales DB
Display Sales Report
Message Receiving
Message Sending
Ending point
[Output Pin]
Exception
Termination Point

Figure 5. 6 Behaviour Scenario Work Process

Behaviour Scenario has the starting point and the end point of the behaviour. The control flow starting from the starting point and terminating at the end point represents the life cycle of Behaviour Scenario. Behaviour Scenario only has one starting point. It is also a parameterised InputPin, which receives the input parameters to instantiate Behaviour Scenario. Behaviour Scenario has a unique end point. It is also an OutputPin, which returns the behaviour results. Besides returning the normal result, Behaviour Scenario contains an exception termination point. The structure makes Behaviour Scenario be able to describe not only behaviours, but also coordination operations, which is transformed into actions in the form of operations. So that it can describe the behaviours at the higher level.

The nesting design of Behaviour Scenario makes the behaviour model be able to describe system objectives with different granularities via hierarchy, which reflects the up-down modelling idea of refining layer by layer for improvement. At the same time, domain meta-models of the more domains are introduced to solve the domain problems with the larger scale. In the nesting hierarchy of Behaviour Scenario, Behaviour Scenario at the high level represents the description of the problem domain in a sense, which can carry out domain-specific application modelling at the more abstract level. Behaviour Scenario at the lower level analyses and refines the specific business objectives, and makes the corresponding model constraints and action specifications, including the necessary behaviour details and behaviour logic. It is able to be used for the complex meta-modelling.

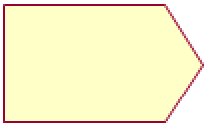
The consecutive executed control flow can be disposed better by Behaviour Scenario. But there are some interrupt processing and parallel processing in many Behaviour Scenarios, for examples, input waiting, asynchronous operation, etc. Therefore, the concepts of message receiving and message sending are introduced into Behaviour Scenario. The interrupted and waiting control flow can be continued to execute by message receiving. A behaviour scenario can cooperates with the parallel processing of entities or other Behaviour Scenarios by message sending.

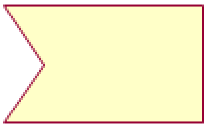
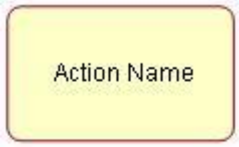
5.2.2 Primary Meta-Model of Behaviour Scenario

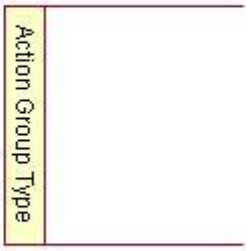
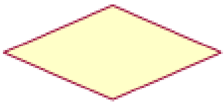
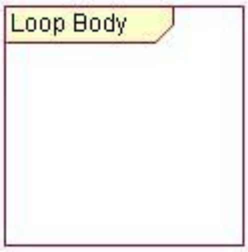
The primary meta-model of behaviour scenario can be constructed by XDML. It describes a series of actions which implement the system behaviours. It is used to illustrate interactions and collaborations among domain objects for an executing process of the specific system objective at implementation time. At the same time, it defines the control flow, the data flow, Action and behaviour execution context accurately by cooperating with AS&MC syntax. The primary meta-model of behaviour scenario is also the foundation of the extension mechanism based on Behaviour Scenario. The more complex domain-specific meta-model can be derived from the extension based on the primary meta-model of behaviour scenario in GME. In the essence, the primary meta-model of behaviour scenario is also a DSL, which is the modelling language that is used for describing the primary behaviour scenario. Its model elements represent the primary behaviour semantics of Behaviour Scenario. Compared to other xDSM meta-models, the behaviour semantics of the primary meta-model of behaviour scenario is implicit, and can be understood by GME and DSMEI. The behaviour semantics of other xDSM meta-models are constructed on the foundation of the primary meta-model of behaviour scenario. The modelling elements of the primary meta-model of behaviour scenario are extracted and organised from the behaviour elements of Behaviour Scenario, which involves:

Table 5. 1 Elements of the Primary Meta-Model of Behaviour Scenario

Starting Point Entity	○	The control flow and the data flow of BS start from Starting Point. It is the starting point of the life cycle. It involves Input Pin, by which the

		parameter information is received from the external to instantiate BS. A behaviour scenario has only one starting point.
Return Point Entity	◎	BS returns the output parameters from Return Point to the external. It involves Output Pin. The return point does not affect the life cycle of BS. A behaviour scenario can have several return points, and all the return points have the consistent Output Pin.
End Point Entity	⊗	The control flow and the data flow of BS terminate at End Point. It is the end point of its life cycle. It involves Output Pin and returns output parameters to the external. A BS contains several end points, and all the end points and return points have the consistent Output Pin. BS implicates the exception termination point. In the cases of absence of exception processing, BS will returns the exception information as the output when an exception happens.
Message Sending Entity		Message Sending Entity provides an asynchronous action execution mechanism. It calls an operation asynchronously to execute the action.

		It encapsulates the input parameters of the operation as a message to send without the return value.
Message Receiving Entity		Message Receiving Entity receives messages from the external of BS and gets the encapsulated input parameters from the received messages. When the control flow executes at the point of Message Receiving, BS will be interrupted and into a dormant state but does not affect behaviour execution context until a particular message arriving. The reason is that the execution and transfer of the action is carried out automatically. But the transfer of the action in the real system is usually triggered by the external information or event, especially in the process modelling of the behaviour with large granularity at high-level.
Action Entity		Action Entity is the most primary action unit, which includes a custom active operation described by AS&MC syntax. It contains action specifications corresponding to the action name. When the control flow passes the action entity, the operation

		is triggered and the action is executed.
Action Group Entity		Several actions are organised by Action Group to build up an execution unit. It does not affect the execution sequences of actions. The type of the action group determines the group action of the execution unit, including: ➢ Transaction of Action Group； ➢ Retry of Action Group； ➢ Exception Catch of Action Group.
Judgement Entity		Judgement Entity contains Boolean expression. It represents an optional path based on the expression. A branch of the control flow is produced in terms of the computation result of the expression, which corresponds to the action sequences which is matched or unmatched the Boolean expression.
Loop Entity		A loop body is contained in Loop Entity. The action sequences of the loop entity element are executed looply under the control of the loop body. The loop body can support these loop structures: For, While and Foreach.

Sequential Relationship Associated Element	——>	All the entity modelling elements are associated by the sequential Relationship. It represents the sequence of the action execution: the next action will start-up in turn after the previous action is completed to complete the transfer of actions.
Exception Relationship Associated Element	⊕——>	Exception Relationship is associated between an action unit (action or action group) and the exception handling action. The control flow will be directed to the exception handling action when the exception comes to the associated action unit. The exception handling action can be a null action which will shield exceptions and return.
Structural Relationship Associated Element	- - - - - - - ->	The entity modelling elements are associated by Structural Relationship. It is unrelated to the concrete behaviour logic and expresses the structural static association.

5.3 xDSM Meta-Modelling Framework

It is more suitable to use domain concepts to construct meta-models for a definite scale of specific domain, rather than to construct meta-models for the larger scale specific domain. On one hand, if the definition granularity of the meta-model element that is corresponding to the domain concept is larger, the contents contained within the element will be more, complex and abstract, and it is more difficult to correspond to the system realisation in addition. On the other hand, if the definition granularity of the meta-model element that is corresponding to the domain concept is smaller, the meta-model will be too complex and hard to be defined and used. The experiences tell us that the larger scale specific domain may involve several smaller scale subordinate specific domains. For example, the domain of archives management and the domain of each business system are involved in the domain of Office Automation. It is necessary for domain-specific meta-modelling to extend the scope covered by the specific domain in breadth, and define the hierarchy according to the domain scale in depth. Especially for xDSM, its executability needs to be ensured from the meta-model layer. So the meta-modelling framework which is scalable, hierarchical and defined accurately is required.

The four modelling levels of OMG are to make people understand the actual contents of models and meta-models better. The relationship of instantiation is its core. As long as each element has its own upper classified meta-element, the meta-data can be accessed via the meta-element, so that any model can be created and any system can be described [47]. In the framework of OMG, meta-models and models both are the relative model concepts based on instantiation. The instantiation relationship determines which abstract levels the model is at. This is a kind of static description relationship. In the dynamic behaviour modelling, there is a relationship as well that to describe the meta-model with the executable application model. That relationship is a dynamic description relationship. In DSM way, it will be provided to modellers in the forms of domain meta-models and application models. Starting from the primary meta-model of behaviour scenario described by XDML, and using application modelling for the meta-model and the method of meta-level promotion, the new meta-model can be created continuously, and the new application model can be created with the new meta-model. This approach develops the domain modelling concepts and its behaviours, rather than carries out the promotion at the abstract level of instantiation. The approach realises the extension mechanism of xDSM meta-model, as shown in the following figure.

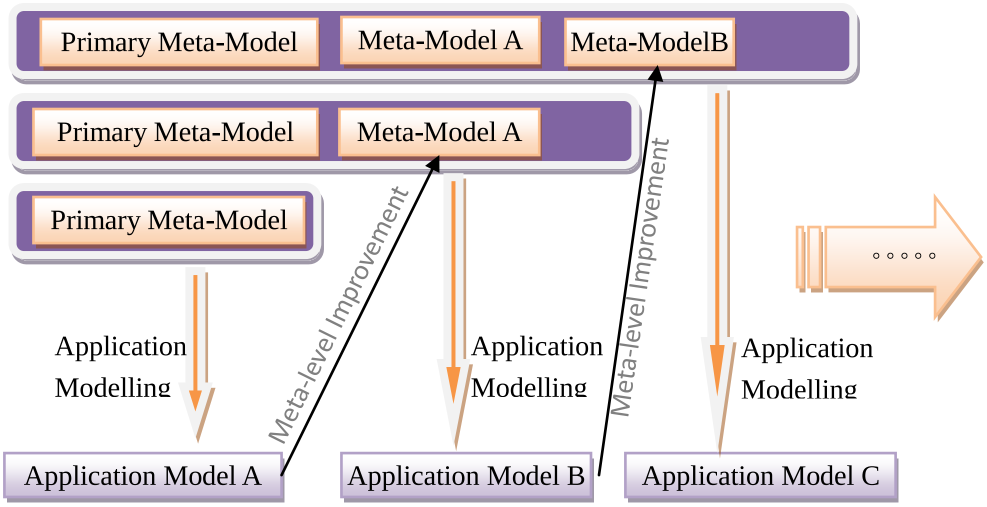


Figure 5. 7 The Extension Mechanism of xDSM Meta-Model

The extension mechanism of xDSM meta-model is the core of the meta-modelling framework of xDSM. The primary meta-model of behaviour scenario is the starting point of the extension mechanism. The behaviour modelling capability of the application model is determined by the accurate behaviour characteristics of the extension mechanism. At the same time, the executability of the application model is guaranteed by the accurate definition of AS&MC syntax. In the process, the primary meta-model of behaviour scenario is used to construct the application model A for the specific domain that is described by the meta-model A, as well as define and add the meta-model A by the way of meta-level improvement to extend the primary meta-model of behaviour scenario. If the larger scale specific domain involves the specific domain described by the meta-model A, the extended the primary meta-model of behaviour scenario which involves the primary meta-model and the meta-model A can be used to construct the application model B for the specific domain that is described by the meta-model B, as well as define and add the meta-model B by the way of meta-level improvement to extend the primary meta-model of behaviour scenario once again. So the extension mechanism of xDSM meta-model can create the large scale domain meta-model incrementally through reusing the subordinate domain meta-model. This is a process of behaviour description, detail encapsulation and constraints. From the perspective of the domain-specific modelling, on one hand, the domain concepts with higher abstract degree that is corresponded to the large-grained meta-model can be created; on the other hand, the xDSM meta-model defined accurately can be constructed on the basis of the subordinate domain meta-models.

The xDSM meta-modelling framework cannot exist alone, and it requires the support of GME to construct xDSM meta-model, as shown in Figure 5.8.

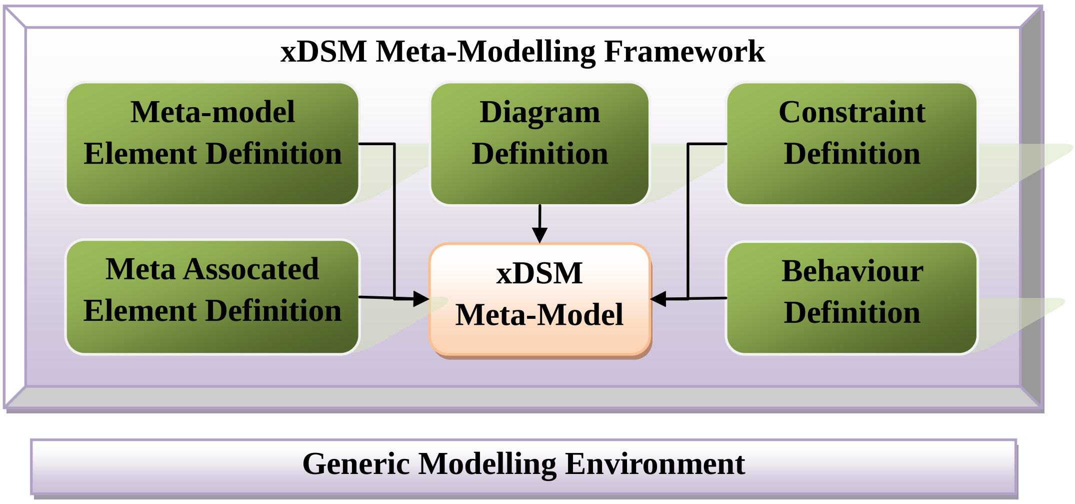

xDSM Meta-Modelling Framework
Meta-model Element Definition
Diagram Definition
Constraint Definition
Meta Assocated Element Definition
xDSM Meta-Model
Behaviour Definition
Generic Modelling Environment

Figure 5. 8 xDSM Meta-Modelling Framework

GME is the implementation environment of xDSM meta-modelling framework. GME supports xDSM meta-modelling to define the meta-model elements, behaviours, constraints and diagrams. At the same time, GME also supports the extension mechanism of xDSM meta-model to describe the behaviours of the established meta-models through application modelling according to the established meta-model. It involves:

- The definition of meta-model element: The domain entity is defined by domain analysis, including attributes, operations (active operations and passive operations), events (modelling time events and runtime events), constraints (invariants) and diagrams.

- The definition of meta-associated element: The relationship is defined by domain analysis. It is a binary relationship to connect the meta-model elements, including attributes, operations (active operations), events (modelling time events), constraints (invariants), diagrams and relationship roles.
- The definition of diagram: The diagram definition is the basis of visual modelling. It specifies visual element for each domain element. While the model elements of xDSM meta-model are registered into GME, the visual application model instances of the meta-model element or the meta-associated element can be constructed with the diagram definition.
- The definition of constraint: The model constraints are defined by AS&MC syntax. It uses meta-data defined by the meta-model to define the pre-conditions, post-conditions and invariants of the constituent parts of the model.
- The definition of behaviour: The behaviour definition of the meta-model is attached to the meta-model elements and the meta-associated elements, including events (modelling time events and runtime events) and operations (active operations and passive operations). All the behaviour definitions will be transformed into behaviour scenarios of the application modelling to construct the behaviours of the meta-model element based on the primary meta-models.

5.4 Summary

xDSM is the core of MDD. The keys to making xDSM models executable are the accuracy and integrality of model, and behaviour modelling. They all are built based on domain-specific meta-modelling.

- Domain-specific meta-modelling is an approach of the systematic model abstract. The abstract is able to reduce the complexity of models and modelling language while it is used to describe system characteristics and maintain the validity of model. xDSM modelling process is divided into meta-modelling phase and application modelling phase, while the roles of modellers are separated at the same time.

- Behaviours are the main expression of system objectives. A series of actions are executed in software specification to achieve system objectives. To eextract the behaviour model corresponding to system objectives and to describe system objectives with behaviours are the keys to the problem-oriented modelling.

- The integrality of xDSM is a subjective and dynamic concept. It requires that end users, domain experts and technical experts work together to construct the complete xDSM which can achieve the domain-specific system objectives. It also requires the overall integrity from xDSM meta-model, xDSM application model to domain-specific supporting services.

- The accuracy of xDSM is based upon xDSM meta-modelling. It integrates the collaborative process of xDSM meta-model and DSMEI. Both of them are complemented and collaborate with each other to realise system objectives, reduce the model complexity, and construct the executable model with sufficient accuracy.

In this chapter, DSM method is employed to build xDSM models. Behaviour Scenario is proposed as the core of behaviour modelling to describe system behaviours according to system objectives by decoupling behaviour logic and computational logic. And the xDSM meta-modelling framework is proposed to construct and assemble xDSM meta-models. The extension mechanism of xDSM meta-model, which is a round trip from meta-models to application models, is proposed to extend xDSM meta-model by the way of using application modelling for the meta-model and the method of meta-level promotion.

Chapter 6

eXecutable Domain-Specific Meta-Modelling Language

A model is a description and specification of the functionalities, structures, behaviours and context of a system. A model needs to be described by a well-defined language. A well-defined language is a language with the strict form (syntax) and meaning (semantics), and can be interpreted and understood by the computer automatically [79]. XDML (eXecutable Domain-specific Meta-modelling Language) is a meta-modelling language which is designed for domain-specific modelling. It is used by GME to support xDSM meta-modelling and application modelling as the description language. Namely, XDML supports the description and construction of xDSM meta-model as well as xDSM application model.

6.1 Introduction

6.1.1 Structure of Modelling Language

The design of modelling language involves syntax design and semantics design. The syntax design involves the design of abstract syntax which is independent on the expression of modelling language and the concrete syntax which is associated with the concrete expression. The concrete syntax is the concrete expression of the abstract syntax. The concrete syntax is generally divided into two kinds: the textual syntax using texts to express; and the graphical syntax using graphics to express. The semantics is used to express the meaning of the concepts which is described by the abstract syntax in modelling language. The well-understanding of semantics of modelling concepts leads to that the modellers can understand and use the modelling concepts correctly. The concept set, that contains the modelling concepts understood and used accurately by the modellers, is the semantics domain of the modelling language. Semantics are also the mapping relationship from the modelling concepts to the concepts of semantic domain, as shown in Figure 6.1.

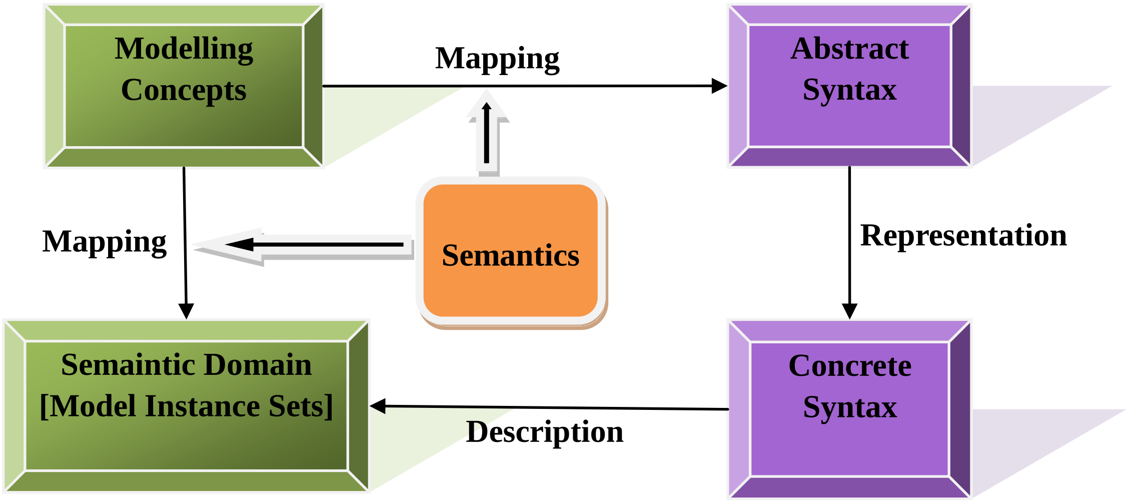


Figure 6. 1 Structure of Modelling Language

An abstract syntax describes the concepts and the relationships among the concepts of the modelling language [17]. To design a modelling language, besides to identify and modelling the concepts of the description language, it is also required to define some rules for the abstract syntax to judge whether the model which is described by the modelling language is legal or not. Those rules will guarantee that the model is flexible. The concrete syntax is provided for modellers to express the model concretely. It is the different view of the abstract syntax. A language can have many kinds of concrete syntaxes. The representation of the concrete syntax of the modelling language can be the textual syntax or the graphical syntax. The concrete syntax with the graphical representation is adopted by the majority of the modelling languages, such as UML. There are some modelling languages only having the textual syntax too, such as OCL and QVT.

To design the modelling language, the relationships among abstract syntax, concrete syntax and semantics are compartmentalised into two mappings which are disjoint: one mapping is from modelling concepts (abstract syntax) to concrete syntax; another is from modelling concepts (abstract syntax) to instances (semantic domain). The above partitions make the abstract syntax, the concrete syntax and semantics can be designed with relatively independent way, reduce the coupling between syntax design and semantics design, and improve the efficiency of the modelling language design.

Semantics of a modelling language is different from abstract syntax of the modelling language. In the thesis, the concrete syntax is used to define the structure and the well-formed relationships of the modelling language. It is the prerequisite of semantics definition. Semantics is the specifications of domain objects and their behaviours. The thesis absorbs parts of the static semantics of XMML into XDML. Those static semantics can be understood by GME, for examples, to check whether the model element types are consistent, whether the connection between model elements can be constructed, etc. The dynamic semantics is expressed by DSMEI collaborated with web services. Thereby, the modelling language can keep the platform independence and the linguistic homogeneity. The focus of the thesis is to analyse and design the behaviour semantics.

6.1.2 XDML Architecture

For enhancing the accuracy of models and the ability of the behaviour modelling in MDA system, OMG issued UML 2.0 which integrates action semantics [35] to improve the ability of the behaviour modelling, and uses OCL to enhance the ability of the accurate model description in MDA system. And ASL (Action Specification Language) is also introduced into xUML to define the system actions in detail. The ultimate goal of above all is to make the behaviour modelling more accurately. UML, OCL and ASL are overlapped in semantics. A part of the abstract syntax of OCL is introduced from the abstract syntax of UML 2.0, especially the introduction of action semantics [90]. ASL is consistent with the action semantics of UML [75]. The coexistence of several sets of abstract syntax of several languages makes it needs a lot of correspondence and references among those languages, and depends on the cohesion of the model reflection interfaces, so as to make the whole syntax architecture huge and complex.

The core of xDSM is the complete and accurate behaviour modelling, with the well-defined behaviour semantics, the accurate model constraints and action specifications as its necessary conditions. XDML is extended based on the semantics of the visual meta-modelling language – XMML. It integrates the well-defined behaviour semantics, supports the domain-specific behaviour modelling adequately, and constructs the concrete syntax of XDML based on XML meta-language. It constructs the textual concrete syntax of AS&MC (Action Specifications and Model Constraints) based on the behaviour semantics of XDML to provide the accurate definition for the dynamic behaviour of models, as shown in Figure 6.2.

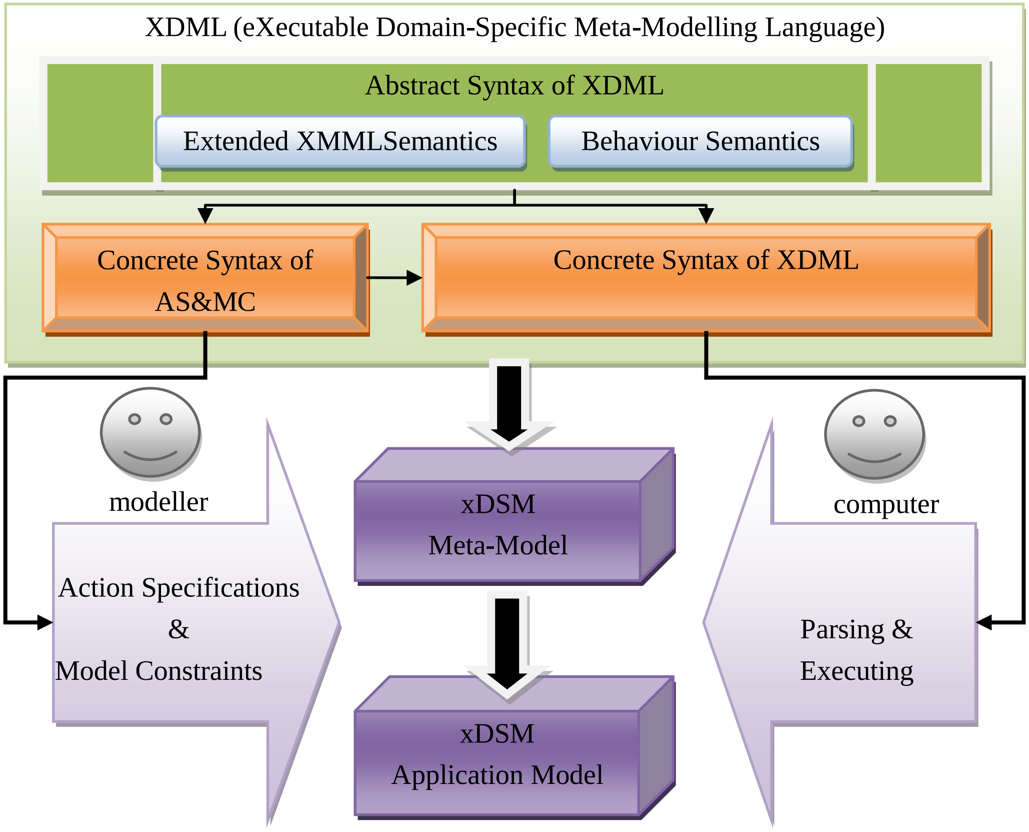


Figure 6. 2 XDML Architecture and Work Process

XDML is the basis of constructing xDSM meta-model and xDSM application model. Model constraints and action specifications are required to define models precisely while behaviour modelling is being carried out accurately. The idea is as follows:

1) The abstract syntax of XDML defines behaviour semantics based on the extended semantics of XMML.

XDML supports the description and construction of xDSM. The visual meta-modelling language -- XMML provides domain meta-modelling and domain application modelling with the complete and valid supports in GME. From the perspective of the static visual modelling, XMML can define and extend the domain concepts completely which are required by domain models. XDML is constructed on

XMML, and extended the XMML semantics for the executable domain-specific model. Behaviour semantics of XDML is defined to meet the requirement of behaviour modelling and to endow the model with the executable behaviour semantics. The abstract syntax of XDML is required to meet the requirement of that model constraints and action specifications is defined precisely, and describe the details of models. What differs from UML language system is that the abstract syntax of XDML is small and precise. Its abstract syntax should be the minimal set to meet the requirements of the construction of the executable domain-specific model.

2) Both AS&MC syntax and the concrete syntax of XDML use the unified abstract syntax.

The concrete syntax of XDML is the concrete expression of the abstract syntax of XDML. AS&MC syntax also uses the abstract syntax of XDML. On one hand, the behaviour semantics involved in XDML is the core of the description of model behaviours, and can support the behaviour requirements of AS&MC. On the other hand, the abstract syntax of XDML provides the way to access and control model elements for AS&MC. AS&MC syntax can enhance the ability of model description. With the unified abstract syntax, semantics is clearer and simpler to avoid duplication and confliction.

3) AS&MC Syntax

Behaviour modelling of UML is too simplex to describe the details of model behaviours clearly. And the complex behaviours lead to that the multi-tier behaviour models have to be used and refined to meet the requirement of complementing the necessary behaviour process. It is very important to provide the unambiguous, accurate and legible AS&MC Syntax for the model details based on behaviour modelling. Action specifications express the action details in a clear and accurate way. At the same time, model constraints provide the accurate constraints (semantics conditions or restrictions) in the modelling process to improve the model description ability. So the accuracy of xDSM can be improved significantly. The abstract of models decides that models are used to describe the system at a certain abstract level. The granularity of the model element description is relatively large. AS&MC syntax can complement the necessary details and constraints, and define models accurately while maintaining the abstract of models. AS&MC is described based on EBNF (Extended Backus-Naur Form), which is used by modellers to describe the details of models. It is similar to using the advanced language, with the friendly user interface and easier to understand.

4) Concrete Syntax of XDML

The concrete syntax of XDML is the concrete expression of the abstract syntax of XDML, and also the integrated expression. XDML is based on XML meta-language, which describes and builds xDSM meta-model and xDSM application model by GME. It involves domain objects, relationships, constraints and behaviour processes. The concrete syntax of XDML is a computer-oriented and textual concrete syntax. It can be identified and displayed by GME, as well as parsed and executed by DSMEI. The concrete syntax of XDML includes all the information described by xDSM and is responsible for models' physical storage. AS&MC syntax will be translated into the concrete syntax of XDML ultimately so as to be handled by DSMEI. That requires algorithms to support the translation from AS&MC syntax to the concrete syntax of XDML.

5) Layered Architecture of XDML

The layered architecture of XDML is divided into three layers: XDML is at meta-modelling language layer, which is used to create the meta-modelling element for the xDSM meta-model.

xDSM meta-model is at meta-model layer. The semantics of XDML is independent of the specific domain. It is an abstract of domain concepts and business processes, and can be used to define the extracted concepts, business and rules from the specific domain. At the same time, xDSM meta-model is the specifications of DSL, which is used to describe and characterise the modelling language.

xDSM application model is at application modelling layer. The definitions of xDSM meta-model are used to construct and assemble the concrete application model of xDSM. AS&MC syntax which is created from XDML at the meta-modelling language layer can be used to accurately define xDSM meta-model and xDSM application model. They will be reflected as the concrete syntax of XDML eventually.

6.1.3 Design Targets

XDML is a domain-specific meta-modelling language which is designed for constructing xDSM. xDSM is based on domain-specific modelling. It is required to provide the support of the complete description language for the domain-specific modelling process, including xDSM meta-modelling and xDSM application modelling. XDML is used to describe xDSM meta-model and xDSM application model accurately, and make xDSM application model executed by DSMEI ultimately. To provide the accurate and complete description ability for xDSM is the design objective of XDML. It is reflected mainly on the two following aspects:

1) For Domain-Specific Modelling:

- ➢ Be able to describe domain concepts, terms in the domain-specific problem domain;
- ➢ Be able to describe the attributes, behaviours and events of domain objects;
- ➢ Be able to describe domain-specific business rules and constraints;

- Be able to describe domain-specific business process.

2) For Behaviour Modelling:

- Be able to modelling the system behaviour accurately;
- Be able to describe action specifications and model constraints accurately;
- Be able to describe data flows and control flows of the system behaviours accurately.

6.2 XDML Abstract Syntax

Constructing abstract syntax is the nature of constructing a modelling language. Abstract syntax of XDML defines a series of model concepts, relationships, and the rule sets which links theses concepts to construct models for xDSM. The abstract syntax of XDML defines the behaviour semantics on the basis of the semantics of the extended XMML to provide the ability of accurate behaviour modelling. Abstract syntax can be divided into many language units. A language unit includes a tightly-coupled abstract syntax concept set. Abstract syntax of XDML is composed of the extended XMML language unit and Behaviour Language Unit, as shown in Figure 6.3.

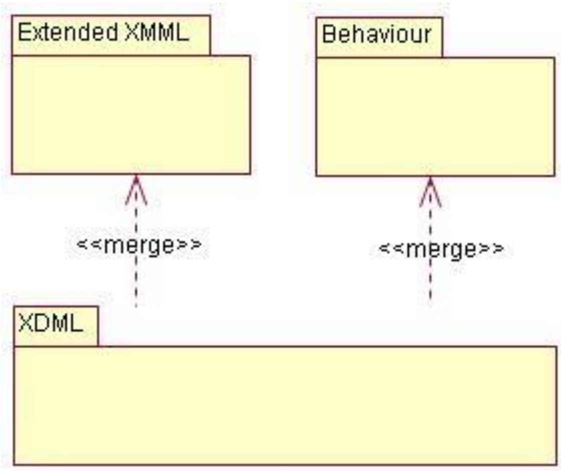


Figure 6. 3 Package Structure of XDML Abstract Syntax

6.2.1 The Extended XMML Language Unit

XMML, the visual meta-modelling language, is the domain-specific meta-modelling language designed for the realisation of the domain-specific modelling [133]. XMML can provide the domain meta-modelling and the domain application modelling with complete and effective support in GME. It is able to completely define and extend the domain concepts which are required by the domain model from the perspective of static visual modelling. The extended XMML language unit includes the abstract syntax concepts of XMML, introduces Behaviour, Action and Constraints, as well as the behaviour concepts that supports the definition of domain rules and domain elements. The extended XMML language unit is the basis of the domain-specific behaviour modelling, as shown in Figure 6.4. The part marked in the black line frame is the task of this thesis for extending the XMML language to provide the ability of the accurate behaviour modelling.

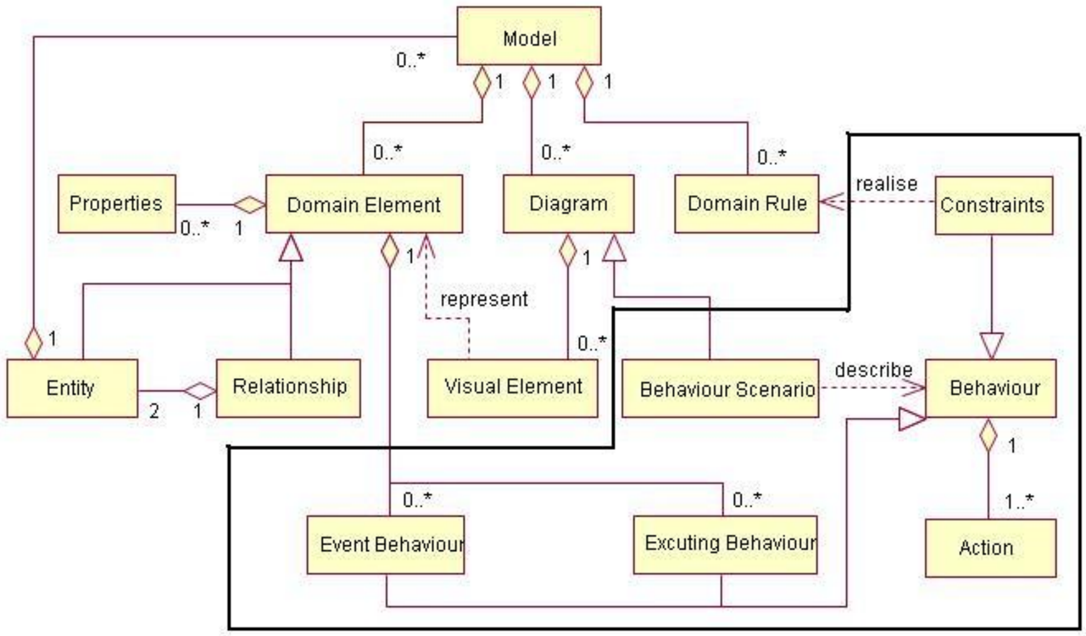


Figure 6. 4 Abstract Syntax of the Extended XMML Language Unit

The following concepts are included in the extended XMML language unit:

1. Model: Model is the descriptions and specifications about software functions, structure, behaviour and its environment. A modelling type or a solution to a domain-specific problem can be expressed as a model. Model represents domain concepts, their relationships, and behaviours, constraints and configuration effecting on the domain elements. Model is composed of Domain Elements, Diagrams and Domain Rules.

2. Domain Element, Entity and Relationship: the main domain concepts of the specific domain are mapped to Domain Elements. Each Domain Element is composed of Properties, Event Behaviour and Executing Behaviour. In the visual modelling process, Visual Element represents the visual design of Domain Element. In the extended XMML language, Domain Elements are derived into Domain Entity and Relationship. Domain Entity is used to represent the types of various entity modelling elements of the domain. Domain Entity will be instantiated as various concrete entity objects in the domain modelling.

Relationship expresses the binary relationship that exists between entities. It will be instantiated as the associations between varieties of Domain Elements.

3. Diagram and Visual Element: Diagram is used to represent an aspect and a part of a model. Diagram includes the View information of an aspect or a part of the model. Visual Element is used to represent Domain Element in Diagram, and define the visual design of Domain Element. A Domain Element can correspond to more Visual Elements, which means that a Domain Element can show different graphical appearances by Visual Element in the different Views.

4. Behaviour, Action and Behaviour Scenario: Behaviour is the core of behaviour modelling, and it reflects the system objective. Behaviour is the direct result of a group of actions of at least a domain concept. Behaviour does not exist by itself. It is attached to Domain Element. Behaviour Scenario is generalised from Diagram, and used to illustrate a series of Actions of Behaviour, describe an execution process of the system, and indicate the interaction and cooperation among domain objects, as well as the implementation of system objectives. Behaviour Scenario is responsible for displaying data of Model, logic relationships and state information in the description way of graphics or text, and unifying Control Flow and Data Flow in the views.

5. Domain Rule and Constraint: Domain Rule is used to characterise the business rules of the application domain and the specifications related to the domain knowledge. When mapped to Model, they are expressed as Model Constraints. Constraint is generalised from Behaviour. It contains a series of Actions and a constraint return. Constraints are used for realising Domain Rules during modelling time and runtime, including pre-conditions, post conditions and infinitives.

6. Event Behaviour and Executing Behaviour: Event Behaviour and Executing

Behaviour are generalised from Behaviours. They are parts of Domain Element, and used to represent the behaviours it own contained. Executing Behaviour is the passive behaviour or cooperative behaviour contained Domain Element. It is an executing set and process of Action. Event Behaviour is a kind of active behaviour, which describes the occurrence set and process of Domain Element.

6.2.2 Hierarchy of Behaviour Language Unit

Behaviour Language Units contains the necessary behaviour semantics that support behaviour modelling, action specifications and model constraints. Behaviour Language Unit is divided into three levels: the first level contains DataType Language Unit and Expression Language Unit, which are used as the behaviour foundation. The second level is Behaviour Core Language Unit which is regard as the core of Behaviour. The third is the Action unit set which is extended from Behaviour Core Language Unit, as shown in Figure 6.5.

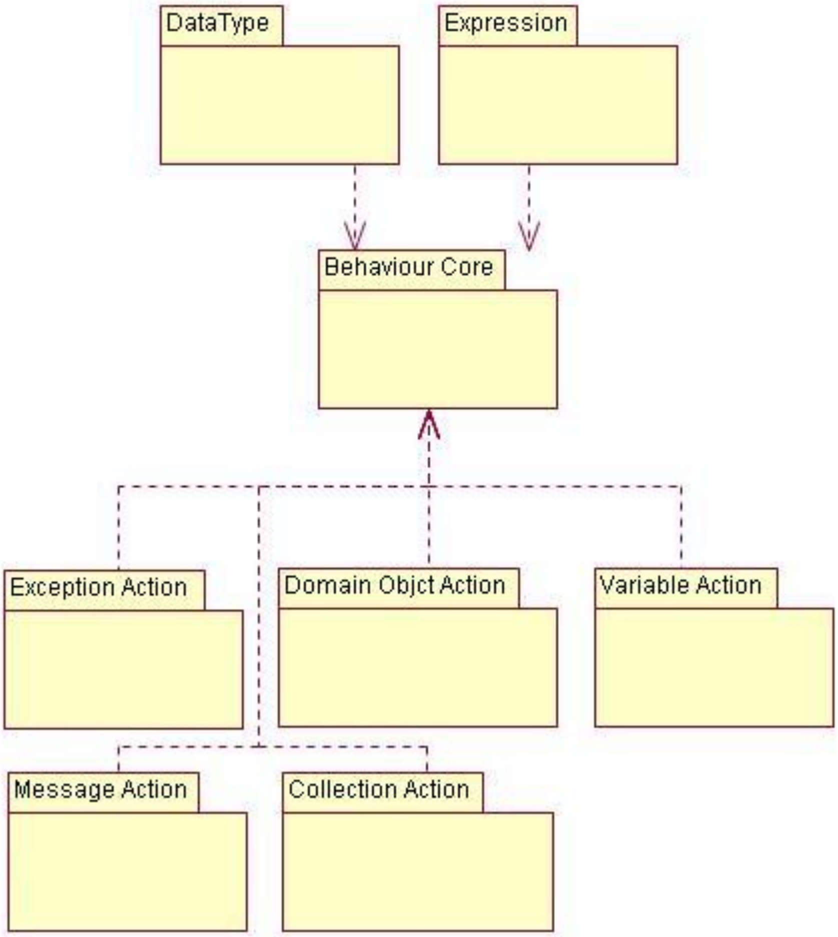


Figure 6. 5 Hierarchy of Behaviour Language Unit

6.2.3 Behaviour Foundation Language Unit

Behaviour Foundation Language Unit supports Behaviour Core Language Unit. It is the necessary condition to describe behaviour and construct the behaviour model. Behaviour Foundation Language Unit contains DataType Language Unit and Expression Language Unit.

1. **DataType Language Unit is shown in the following figure.**

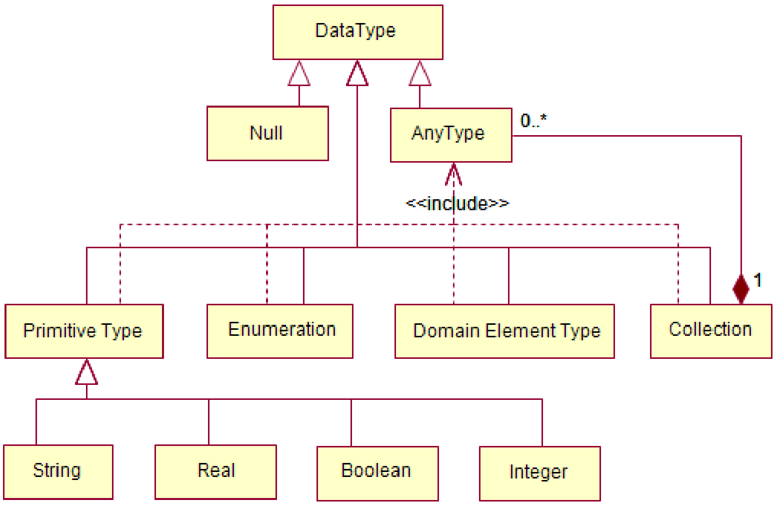


Figure 6. 6 DataType Language Unit

DataType Language Unit describes the data type in XDML language. It is not only the basis of the accurate behaviour modelling, but also the necessary condition of the data flow description. It is used to describe the various related data types of the system behaviours. Various kinds of concrete types are generalised from DataType, including:

- Null and AnyType: Null represents a data type without value. AnyType can denote any valid data type. They are especial data type.
- Primitive Type: it is the basic and the commonly used data type, including String, Integer, Real and Boolean.
- Enumeration: a limited number of identifiers which is used to represent a group of continuous constants.
- Domain Element Type: it is used to describe the complex data type of Domain Element, and coordinate with the reflection interface to access Entity and Relationship within Model.
- Collection: it is a set of the data which have the same type. The element types supported by Collection include all the valid data types, such as Primitive Type, Domain Element Type, etc.

2. Expression Language Unit:

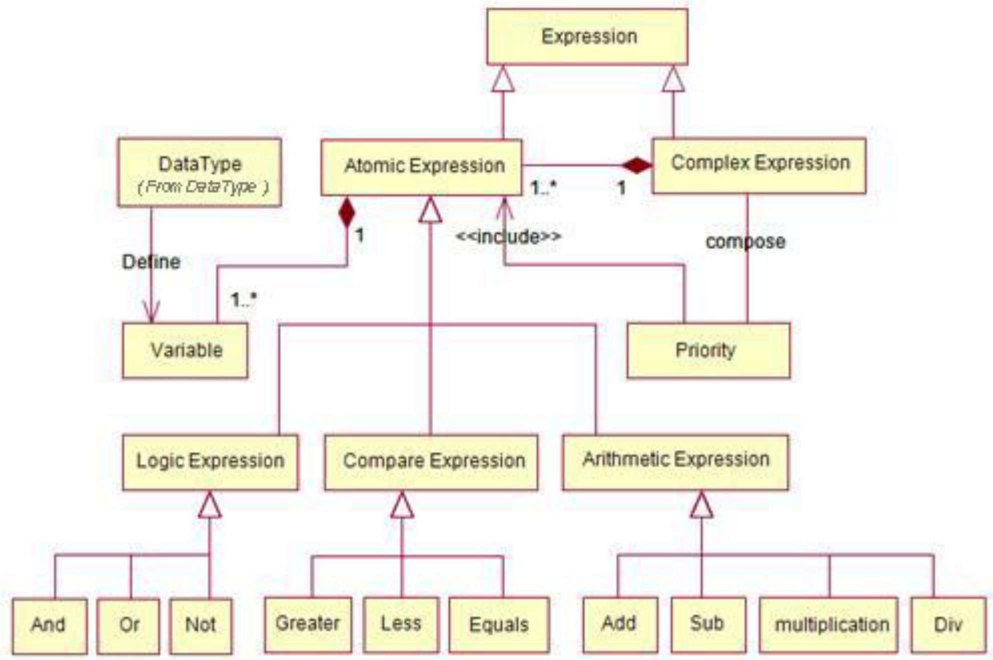


Figure 6. 7 Expression Language Unit

Expression Language Unit describes the expressions of XDML language. It is composed of operators, constants and variables. It can calculate a result of the operations. It is the basis of data computing, and the important part of behaviour modelling for supporting model constraints and computational logic actions. Various kinds of the concrete expressions are generalised from Expression, including:

- Atomic Expression: Atomic Expression is the basic Expression Language Unit. It represents an operation which cannot be subdivided. It may contain a number of operation variables, which is provided with variables, constants, or operations. Atomic Expression is generalised into three basic expressions: Logic Expression provides the logic operations, such as And, Or and Not; Comparative Expression provides Comparative operations, such as Greater, Less and Equals; Arithmetic Expression provides arithmetic operations such as Add, Sub, Multiplication and Division.

- Complex Expression: Complex Expression is the combination of Atomic Expressions. It contains at least one Atomic Expression. Complex Expression carries the composition operation according to the priority of each Atomic Expression to get the computing results of the expression.

6.2.4 Behaviour Core Language Unit

Behaviour Core Language Unit is the core unit to support behaviour modelling in XDML. Based on DataType Language Unit and Expression Language Unit, Behaviour Core Language Unit describes the behaviour structure completely, including Behaviour, Action and Operation. There are many concepts introduced from the extended XMML Language Unit, which relevant to each other.

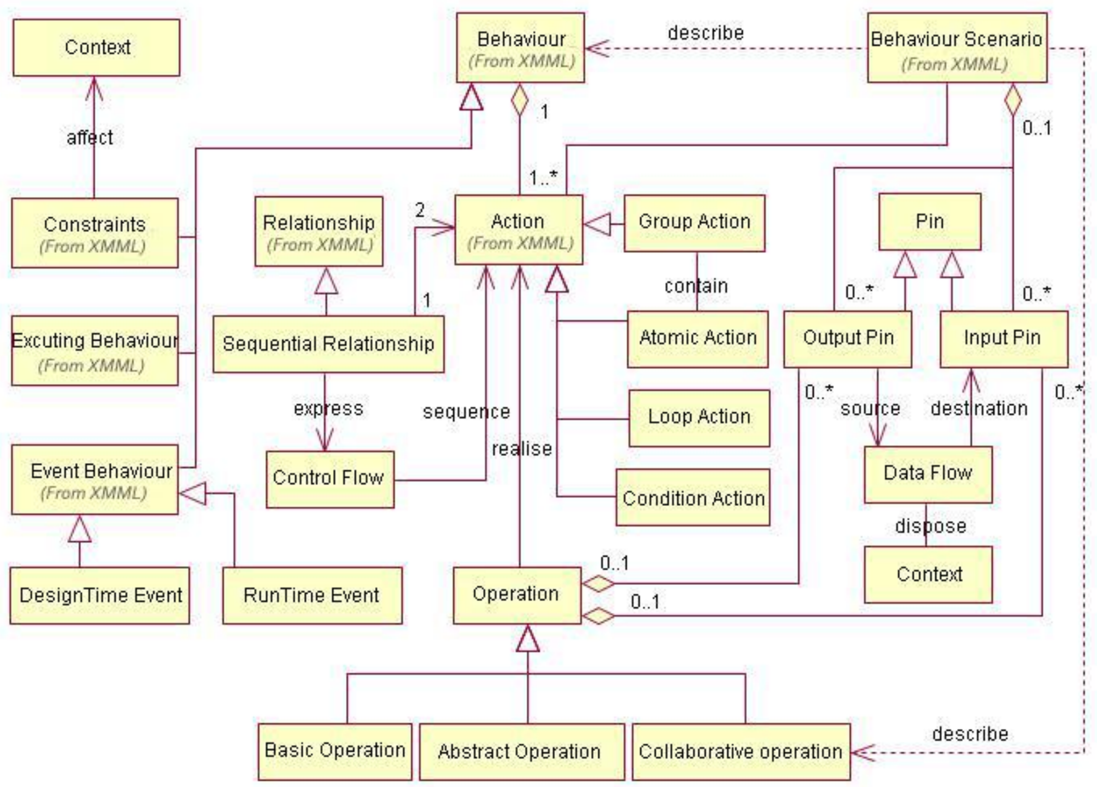


Figure 6. 8 Behaviour Core Language Unit

The following concepts are included in Behaviour Core Language Units:

1. Behaviour and Behaviour Scenario: Behaviour is the core of behaviour modelling. It embodies the system objective. In XDML language, Behaviour Scenario is used to describe the behaviour, and explain the executing process of a series of Actions contained in a behaviour. Behaviour Scenario is expressed as a entirety in the form of Operation, which includes Input Pin and Output Pin. BS can be transformed into a Coordination Operation.

2. Action: Action is the basic unit of behaviour semantics. It represents the state transformation or the handling operation of a system element. Action can be generalised into four kind of basic Actions: Atomic Action, which represents a basic action that cannot be subdivision; Group Action: Atomic Actions are assembled as a group of actions and have some characteristics of the group execution, including basic group actions, transaction actions, retrying actions and exception catching actions; Loop Action, which executes a group of actions circularly under the loop control; Condition Action: a group of actions is executed when the control condition is met, otherwise the loop is quitted or another group of actions is executed.

3. Operation: Operation supports the execution of Action, and realises operation semantics of a behaviour. Operation includes Input Pin and Output Pin, which transforms a group of input into a group of output. The following three kinds of operations are generalised from Operations. Basic Operation, that supports Basic Action and provides more primitive and commonly used operations; Abstract Operation, that is an abstraction of the concrete implement operations and describes the structured interface information of the operation. It corresponds to the concrete implement of the atomic operation services of domain business; Coordination Operations, that is constructed by the way of Behaviour Scenario based on Action. It is used to describe the behaviour logic of coordination operations among the domain concepts.

4. Pin and Data Flow: Pin is both type element and multiplicity element. Namely, the data of Pin is multi-valued, sequential and unique. Pin provides values to Operation as well as gets the returned values from Operation. Accordingly, Pin is generalised into Input Pin and Output Pin. The executing process of a behaviour is from the source of the Output Pin of an Operation, based on Execution Context to deal with the data then transfer to the Input Pin of another Operation. That constitutes a whole data flow. The data flow is not an absolute concept of data pipeline, but can get and assign data clearly from Execution Context and Operation, and support Control Flow working normally.

5. Control Flow: Sequential Relationship represents a sequential execution relationship between actions. It is generalised from Relationship. Sequential Relationship is associated with two actions, and expresses the control flow of action executions. It also defines the order of action executions that are depended on each other: follow-up actions can be executed only after the previous actions have been done.

6.2.5 Action Language Unit

The set of Action Language Unit is constructed based on behaviour modelling foundation of Behaviour Core Language Unit. It extends action semantics by the pertinent generalised Action, and provides more abundant basic Actions to support Behaviour to achieve system objectives. Action Language Unit is the flexible and scalable set of language units. At this stage, it generally includes:

- Language Unit of Exception Action: It is designed for the exception handling actions in the behaviour executing process, including Exception Throwing Action and Exception Catching Action.
- Language Unit of Domain Object Action: It is designed for the actions of domain

objects (such as Entity and Relationship), including Creation Action of domain object, Destroy Action of domain object, and Data Access Action of domain object.

- Language Unit of Variable Action: It is designed for the actions in connection with the data of variable, including Variable Declaration Action, Variable Access Action and Variable Update Action.
- Language Unit of Message Action: It is designed for the actions of the system messages, including Message Sending Action and Message Receiving Action.
- Language Unit of Collection Action: It is designed for the actions of the data type of collection, including Collection Index Action, Collection Traversal Action and Collection Dynamic Setting Action.

6.3 AS&MC Concrete Syntax

AS&MC provides the accurate Action Specifications and Model Constraints for xDSM modelling. With action specifications, what processes happen in an Operation can be declared. Action specifications is able to introduce model elements, operate domain objects and relationships, call the related operation, send message, and describe the behaviour at the abstract level of domain model. The model constraints is able to enhance the description ability to express semantics of the modelling elements in models, so as to accurately define Domain Rules that are expressed by the model during the periods of modelling and running. At the time of domain model running, Domain Rules is also embodied in action specifications, so as to make action specifications and the runtime model constraints integrated. The unified expression of the concrete syntax of Action Specifications and Model Constraints semantics is propitious to describing models collaboratively and accurately.

The thesis uses EBNF (Extended Backus-Naur Form) [127, 110] to describe the concrete syntax of AS&MC. EBNF is a family of meta-syntax notations used to express context-free grammars. It is a formal way to describe computer programming languages and other formal languages. EBNF is developed on the basis of BNF (Backus-Naur Form) [57]. Its expressing ability is the same as BNF, but its structure is simpler and clearer, and easy to use. The basic contents of EBNF are:

1. ::= ： is defined as
2. "..." ： terminals
3. <… > ： nonterminals, represent syntax constituents
4. [...] ： optional items, occur up to once
5. {...} ： repeated options, which can be repeated from 0 to any times
6. | ： parallel options, only one can be chosen from the multiple options
7. (...) ： syntax packet

The concrete syntax of AS&MC is defined as follows.

- **The definition of Syntax Unit and Statement:**

<Constraint> ::=

Constraint <Identifier> <Block> .

<Operation > ::=

<Operation-heading> ";" <Block> .

<Operation-heading> ::=

Operation <Identifier> [<Formal-parameter-list>] [":" <Type-identifier>].

<Formal-parameter-list> ::=

"(" <Formal-parameter-section> { ";" <Formal-parameter-section> } ")" .

<Formal-parameter-section> ::=

<Identifier-list> ":" <Type-identifier> .

<Block> ::=

"{" <Statement-sequence> "}" .

<Statement-sequence> ::=

<Statement> { ";" <Statement> } .

<Statement> ::=

<Simple-statement | <Structured-statement> .

<Simple-statement> ::=

[<Constant-definition> | <Variable-declaration> | <Assignment-statement> | <Operation-statement>] .

<Constant-definition> ::=

Const <Identifier> "=" <Constant> .

<Variable-declaration> ::=

Declare <Identifier-list> ":" <Type> .

<Assignment-statement> ::=

<Variable> ":=" <Expression> .

<Operation-statement> ::=

<Operation-identifier> [<InputPin>] .

<Structured-statement> ::=

<Compound-statement> | <Repetitive-statement> | <If-statement> .

<Repetitive-statement> ::=

<While-statement> | <For-statement> | <Foreach-statement> .

<While-statement> ::=

while <Expression> **do** (<Statement> | <Block>) .

<For-statement ::=

for " ("<Assignment-statement> ";" <Expression> ";" <Assignment-statement> ")" (<Statement> | <Block>) .

<Foreach-statement> .

Foreach " ("<Type-identifier> <Variable-identifier> **in** <Variable> ")" (<Statement> | <Block>) .

<If-statement> ::=

if <Expression> **then** (<Statement> | <Block>) [**else** (<Statement> | <Block>)] .

<Pin> ::=

<InputPin> | <OutputPin> .

<InputPin> ::=

"(" <Actual-parameter> { "," <Actual-parameter> } ")" .

<OutputPin> ::=

<Type> .

<Actual-parameter> ::=

<Expression> | <Variable> | <Actual-Operation> .

<Actual-Operation> ::=

<Operation-identifier> .

- **The definition of Expression:**

<Expression> ::=

<Simple-expression> [<Relational-operator> <Simple-expression>] | <Collection-Query>.

<Simple-expression> ::=

[<Sign>] <Factor> { <Expression-operator> <Factor> } .

<Factor> ::=

<Variable> | <Number> | <String> | **nil** | <Constant-identifier> | <Bound-identifier> | <Function-designator> | "(" <Expression> ")" | **not** <Factor> .

<Relational-operator> ::=

"==" | "<>" | "<" | "<=" | ">" | ">=" .

<Expression-operator> ::=

"+" | "-" | "*" | "/" | **and** | **or** .

<Collection-Query> ::=

<Variable> "(" <Type> "|" <Expression> ")" .

<Variable> ::=

<Entire-variable> | <Component-variable> | <Referenced-variable> .

<Entire-variable> ::=

<Variable-identifier> | <Field-identifier> .

<Component-variable> ::=

<Indexed-variable> | <Field-designator> .

<Indexed-variable> ::=

<Collection-variable> "[" <Element-list> "]" .

<Element-list> ::=

[<Expression> { "," <Expression> }] .

<Field-designator> ::=

<Object-variable> "." <Field-identifier> .

<Operation-designator> ::=

<Operation-identifier> [<InputPin>] .

- **The definition of Type and Assistant Syntax:**

<Type> ::=

<Primitive-type> | <Enumerated-type> | <Collection–type> | <Domainelement-type> | <Type-identifier> .

<Primitive-type> ::=

<String> | <Real> | <Boolean> | <Integer> .

<Enumerated-type> ::=

"(" <Identifier-list> ")" .

<Collection-type> ::=

Collection ["[" <Integer> { "," <Integer> } "]"] **of** <Type> .

<Integer> ::=

```
    <Digit-sequence> .
<Real> ::=
    <Digit-sequence> "." [<Unsigned-digit-sequence> ] |
    <Digit-sequence> .
<Digit-sequence> ::=
    [<Sign> ] <Unsigned-digit-sequence> .
<Unsigned-digit-sequence> ::=
    <Digit> { <Digit> } .
<Sign> ::=
    "+" | "-" .
<String> ::=
    "’" <String-character> { <String-character> } "’" .
<String-character> ::=
    <Any-character> | "’’" .
<Boolean> ::=
    "true" | "false" .
<Constant> ::=
    [<Sign> ] (<Integer> | <Real>) | <String> .
<Domainelement-type> ::=
    <Identifier> .
<Type-identifier> ::=
    <Identifier> .
<Operation-identifier> ::=
    <Identifier >.
<Identifier> ::=
    <Letter> { <Letter> | <Digit> } .
<Identifier-list> ::=
    <Identifier> { "," <Identifier> } .
```

<Comment> ::=

"/*" <Any-character> "*/" .

<Include> ::=

include <String> .

<Letter> ::=

"A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" |
"N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" |
"a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" |
"p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" .

<Digit> ::=

"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .

6.4 XDML Concrete Syntax

The concrete syntax of XDML is defined by XML Schema [94]. XSD (XML Schema Definition) is a W3C standard which is used for the type system called XML Schema which is based on XML. The language used for definition is a kind of XML syntax called XML Schema Definition Language. XML Schema document itself is the validating XML. Compared to the early DTD, XML Schema has the following characteristics, for examples, simpler format, easier to understand and stronger capacity of expression. At the same time, XML Schema is convenient for forming SOM (Schema Object Model) and good for the application to carry out the syntax parsing and validation of the object XML document according to XSD.

XDML is the domain-specific meta-modelling language based on XML meta-language. It employs the unified concrete syntax to describe xDSM meta-model as well as xDSM application model (the detailed definition of XSD refers to Appendix A). XDML is computer-oriented, responsible for model persistence, model visualisation, and the parsing and executing of models. The concrete syntax of XDML is extended on the basis of the concrete syntax of XMML. It supports domain-specific meta-modelling, application modelling as well as model visualisation. The concrete syntax of XDML is described on the following two aspects:

- Domain-specific modelling and model visualisation: They are inherited from the concrete syntax of XMML, and carry out the behaviour extension of the model entities so as to support behaviour modelling.
- Accurate behaviour modelling: it integrates action specifications and model constraints of models from AS&MC.

6.4.1 Domain-Specific Modelling

1. Domain Model

A domain modelling objective or a solution to a domain-specific problem is represented as a domain model. In XDML, the domain model is composed of framework elements (for examples, views, domain entities and relationships) which is necessary for the model, and the related behaviours and their details of the domain modelling objective (for examples, operations, constraints and events). Its concrete syntax structure is shown in Figure 6.9.

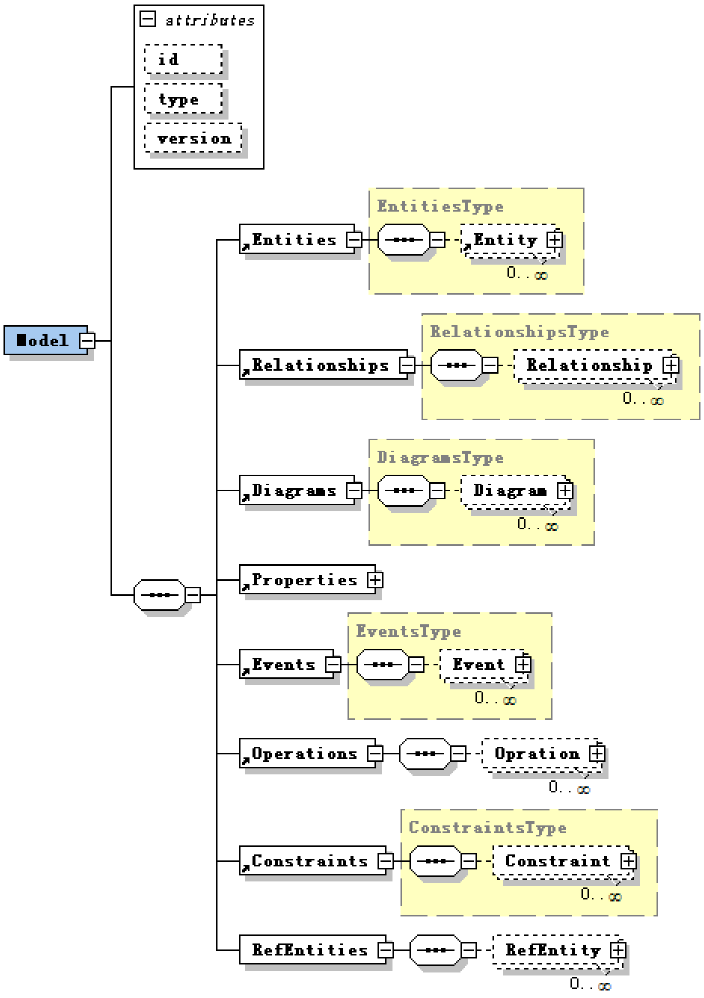


Figure 6. 9 XML Schema Definition of Model

Model is a ComplexType, which involves:

Attributes:

- ID: It is the unique identification of a model, and also the namespaces of model elements, views and operations.
- Type: It is the type of a model.
- Version: It is the version identification of a model.

Child Elements:

- Entities: It is the set of various domain concepts and domain object entities of a model.
- Relationships: It is the set of associations among various entities of a model.
- Diagrams：To show the set of diagrams of a model in the visual way. Diagrams are used to describe Behaviour Scenario visually in XDML.
- Properties：It is the set of various properties information of a model;
- Events: It is the event set involved in a model itself.
- Operations: It is the set of operations at the model level, including abstract operations and behaviour operations with the larger granularity.
- Constraints: It is the set of constraints at the model level.
- RefEntities：It is the set of reference entities that is introduced from the external of a model.

2. Domain Entity

In meta-modelling and application modelling, the domain concepts and the instantiated entity objects are used to represent the content of model entities. Domain Entity is used to represent various entities of modelling elements of the specific domain, and will be instantiated as various concrete entity objects in domain application modelling. Its concrete syntax structure is shown in Figure 6.10.

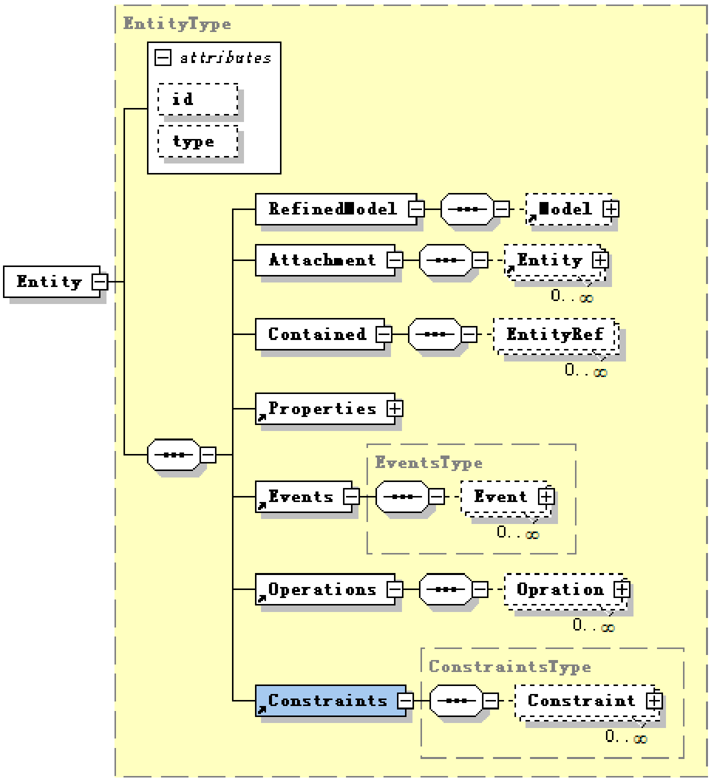


Figure 6. 10 XML Schema Definition of Entity

Entity is a ComplexType, which involves:

Attributes:

- ID: It is the unique identification of an entity.
- Type: It is the type of an entity. It is used to identify its meta-model elements.

Child Elements:

- RefinedModel: It is the refined model contained in an entity. Model is refined further by establishing sub-models.
- Attachment: It is the set of attachable sub-entity objects in an entity. Entity is responsible for the life cycle of sub-entity.
- Contained: It is the set of the referenced entity objects that is contained in an entity. The relationship between them and the entity is loosely coupled.

- Properties: It is the set of various properties information of an entity.
- Events: It is the event set of an entity, including modelling time events and runtime events.
- Operations: It is the set of operations of an entity, and the set of the attached Executing Behaviours of an entity. They are represented as active operations or passive operations.
- Constraints: It is model constraints at the entity level.

3. Relationship

The relationship in Model is used to describe the binary relationship existing between Entities, and establish the association between Entities. The roles played by the connected Entities can be specified in Relationship. The data flow of Entities can be joined and the control flow of behaviour modelling can be embodied in Relationship. Its concrete syntax structure is shown as the following figure.

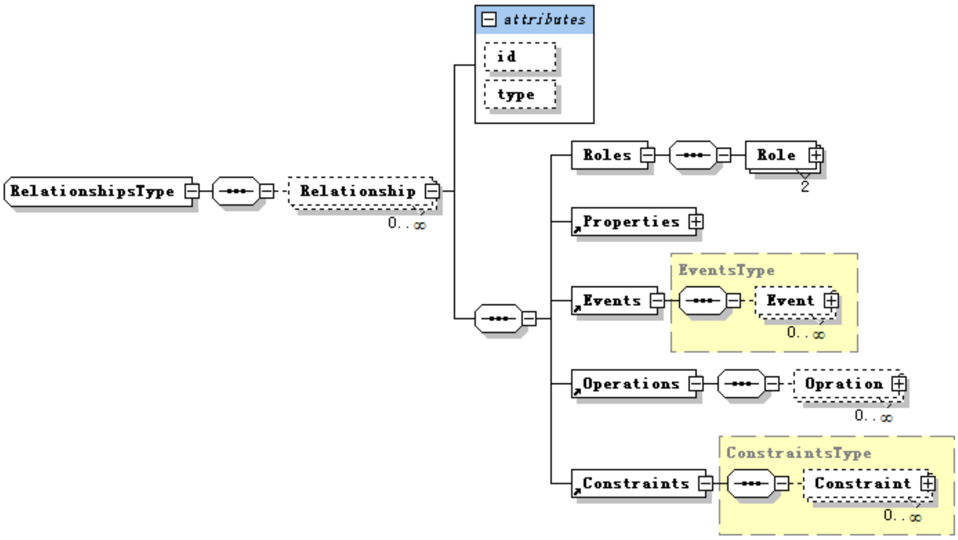


Figure 6. 11 XML Schema Definition of Relationship

Relationship is a ComplexType, which involves:

Attributes:

- ID: It is the unique identification of a relationship.
- Type: It is the type of a relationship. It is used to identify its meta-model elements.

Child Elements:

- Roles: It shows the role information of the two entities which are connected by a relationship. The role embodies the position, effect and identity, etc. of the entity in the binary relationship.
- Properties: It is the set of various properties information of a relationship.
- Events: It is the event set of a relationship, including modelling time events and runtime events.
- Operations: It is the set of operations of a relationship. It is action specifications which connect the entity behaviours, and represented as the active operation.
- Constraints: It is model constraints at the relationship level.

4. Diagram

The visualisation method is used to display the diagrams of models. Diagrams are the visualisation definition of models. VisualElements are corresponded to the modelling elements of the domain model and describe its visualisation information. Diagram is the interactive interface with users; at the same time, it is also a logic unit of the domain model. Its concrete syntax structure is shown as the following figure.

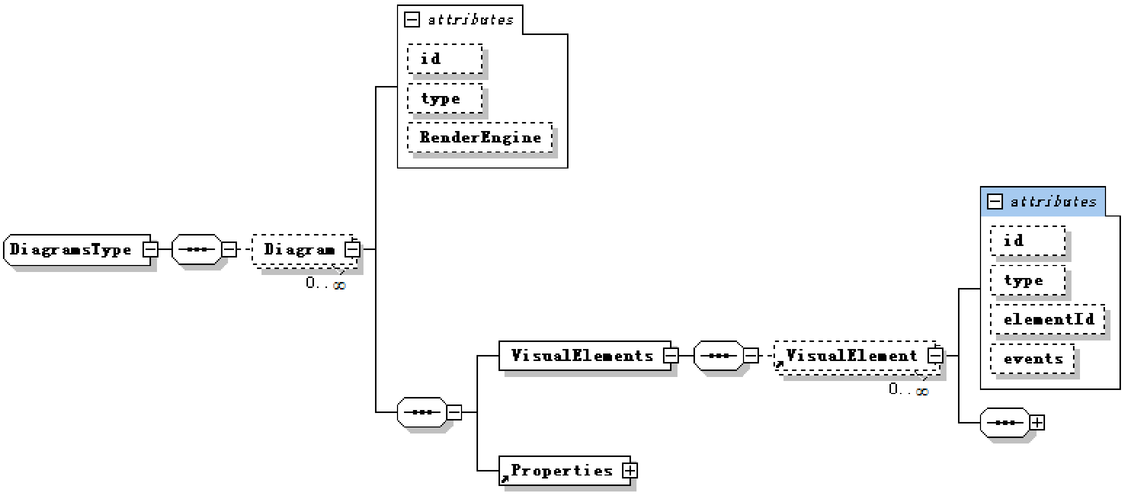


Figure 6. 12 XML Schema Definition of Diagram

Diagram is a ComplexType, which involves:

Attributes:

- ID: It is the unique identification of a diagram.
- Type: It is the type of a diagram, such as Behaviour Scenario.
- RenderEngine：It is the render engine of a diagram.

Child Elements:

- VisualElements：It is the set of the visual information of each modelling element in a diagram. It is associated with the modelling element by its ElementID.
- Properties: It is the set of various property information of a diagram.

6.4.2 Behaviour Modelling

1. Intermediate of AS&MC

AS&MC syntax is used to describe action specifications and model constraints accurately. Its concrete syntax is similar to that of the advanced language. AS&MC syntax is integrated into XDML as the intermediate of AS&MC. The intermediate of AS&MC is the structured representation of AS&MC syntax which is processed by the lexical and syntactic analysis. It is the basis for the computer to understand behaviours and constraints. Its concrete syntax structure is shown as the following figure.

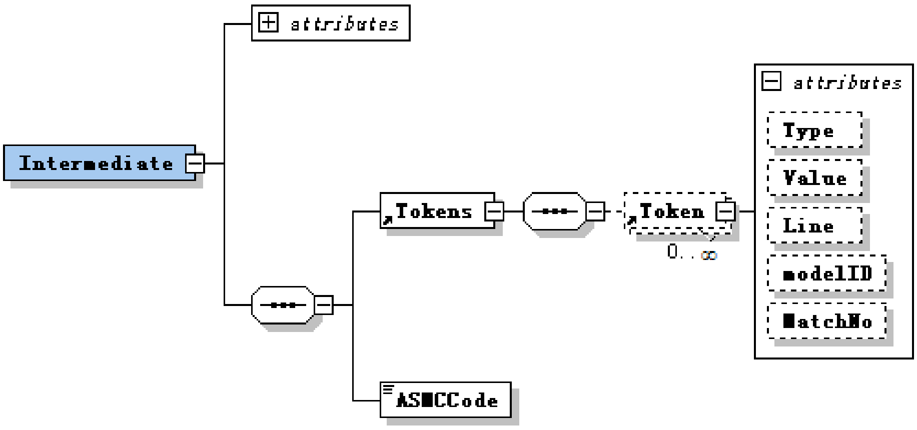


Figure 6. 13 XML Schema Definition of Intermediate

Intermediate is a ComplexType, which involves:

Attributes:

- Type ： It is type, operation or constraint described by Intermediate.

Child Elements:

- Tokens： It is the indivisible logic unit of AS&MC syntax. Token is the structured representation of AS&MC syntax which is processed by the lexical and syntactic analysis. It represents the minimal token unit, which can be used to express the concrete operation or data by the individual or grouped way.

  Each token is composed of the following attributes:

- Type: It is the classification of Tokens, including separators, keywords, identifiers, etc.
- Value：It is the concrete value of a Token.
- Line: It is the number of lines which a Token corresponds to.
- ModelId：It is the identification of the domain model which a Token corresponds to, and also the corresponding namespace.
- MatchNo：It is the sequence number of the paired tokens. It is only valid for the paired tokens.

➢ AS&MC Code：It is the source code of action specifications or model constraints which is described by AS&MC syntax.

2. Operation

Operation is the concrete expression of Executing Behaviour. It is divided into Abstract Operation and Coordination Operation. Coordination Operation can be associated with a Behaviour Scenario. At the same time, Operation can act as the carrier of action specifications and joins Entity. Operation is the main representation of Action and the basic unit of action specifications. Its concrete syntax structure is shown as the following figure.

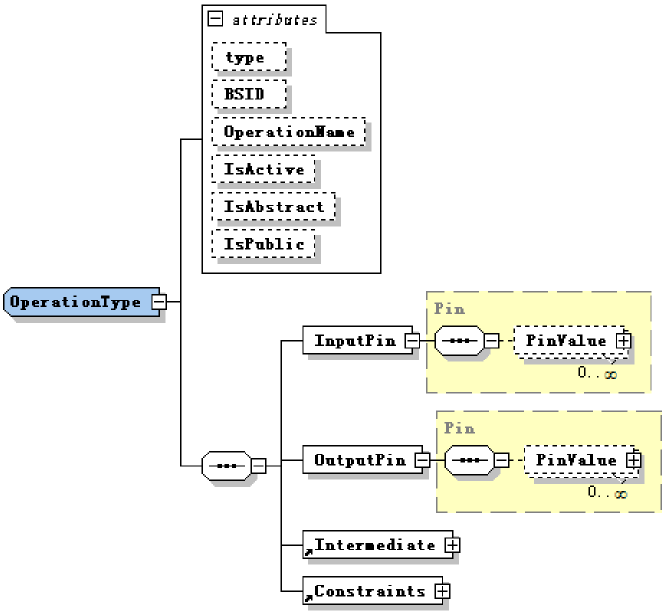


Figure 6. 14 XML Schema Definition of Operation

Operation is a ComplexType, which involves:

Attributes:

- OperationName：It is the name of an operation, and the unique identification of the operation in the model namespace.
- Type : It is the type of an operation, for examples, behaviour operation and action operation.
- BSID: It is the ID of Behaviour Scenario that Coordination Operations correspond to.
- IsActive: It says whether the operation is active operation or not.
- IsAbstract: It says whether the operation is abstract operation or not.
- IsPublic: It says whether the operation is public or not.

Child Elements:

- Input Pin: It is the input pin of an operation. It is an ordered sequence of names and types.
- Output Pin: It is the output pin of an operation.
- Intermediate: It is the intermediate of action specifications described by AS&MC syntax.
- Constraints: It is the set of constraints of an operation, including pre-conditions and post-conditions.

3. Constraint

In the modelling process, the accurate constraints are provided by Constraints to represent domain rules and to enhance the description ability of behaviour modelling. Constraints can be used to represent domain rules at both modelling time and runtime. Model constraints are embodied in GME by the events of modelling elements at modelling time. At runtime, domain rules require to be expressed explicitly, and constraints are realised in the concrete model execution. Its concrete syntax structure is shown as the following figure.

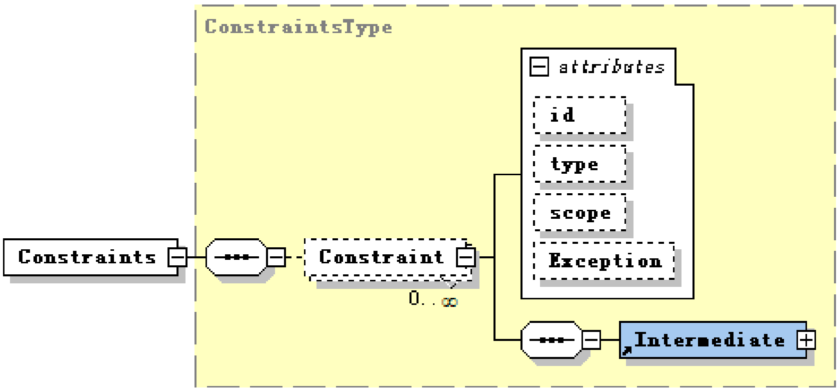


Figure 6. 15 XML Schema Definition of Constraint

Constraint is a ComplexType, which involves:

Attributes:

- ID: It is the unique identification of a constraint.
- Type: It is the type of a constraint, for examples, Pre-Conditions, Post-Conditions or Invariant.
- Scope: It is the effect scope of a constraint, namely, effect domain, which can be attributes or operations of the domain model or the domain entity.
- Exception: It is the exception operation of a constraint, which is a appointed exception handling action when the constraint is not be met;

Child Elements:

- Intermediate: It is the intermediate of model constraints described by AS&MC syntax.

4. Event

Event is the concrete expression of Event Behaviour. It corresponds to Occurrence of domain concepts, which answers to the external messages and executes the corresponding operations. At the same time, Event is also the means to embody model constraints at modelling time in GME. Its concrete syntax structure is shown as the following figure.

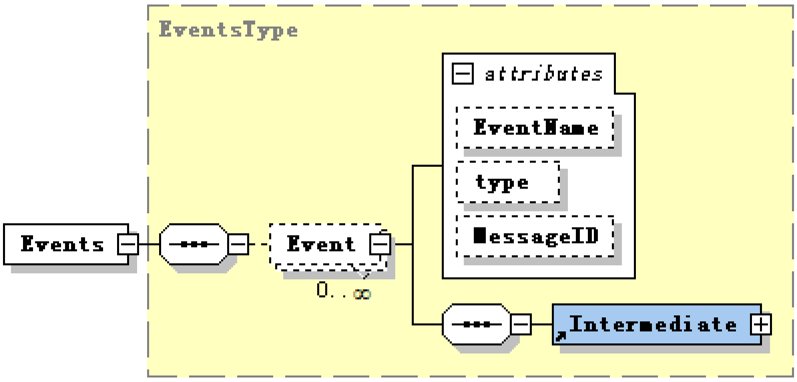


Figure 6. 16 XML Schema Definition of Event

Event is a ComplexType, which involves:

Attributes:

- EventName: It is the name of an event, and the unique identification of the event in the model namespace.
- Type: It is the type of an event, for examples, Event Behaviour and Modelling Constraints Events.
- MessageID: for Event Behaviour, it is the ID identifier of the message which triggers the event.

Child Elements:

- Intermediate: It is the intermediate of model constraints or action specifications described by AS&MC syntax.

6.5 Summary

In this chapter, XDML language is defined to describe xDSM. XDML Language is a meta-modelling language which is designed for DSM. It is used by GME to support xDSM meta-modelling and application modelling as the description language. XDML language is the foundation for the model execution.

XDML language integrates well-defined behaviour semantics to support domain-specific behaviour modelling. The concrete syntax of action specifications and model constraints are built on the basis of behaviour semantics of XDML language, which is used to define behaviour details and behaviour constraints of xDSM meta-model and application model, so as to describe systems in detail and accurately.

Chapter 7

Domain-Specific Model Execution Infrastructure

With the accurate definition of XDML, xDSM application model is built based on xDSM meta-model for a domain-specific application. xDSM application model describes system behaviours accurately and completely, and meets the requirements of MMLs 5. However, only xDSM is impossible to be executed. It must depend on some execution environment to be parsed and executed. DSMEI (Domain-Specific Model Execution Infrastructure) provides the executable environment for xDSM application model. DSMEI is responsible for parsing behaviour semantics of xDSM application model, transforming them into the operational sequence with accurate semantics, and executing these operations to achieve system objective. DSMEI integrates domain framework and combines AGOS to provide software functional entities for the virtual operations, thereby which makes xDSM application model become the executable software product in DSMEI.

7.1 Architecture

With the development of network technology, software platform has been evolving from traditional stand-alone, closed, static runtime environment into varied, open, dynamic network runtime environment gradually. DSMEI is a software platform within network environment, as well as the execution environment for xDSM models. DSMEI takes the accurate and integrated behaviour logic of xDSM as the core and AGOS as software functional entities, so as to transform xDSM application model into the service-oriented domain-specific application. The service-oriented application is a kind of software system in network environment. It is a natural extension of the traditional software structure [128].

DSMEI parses and executes xDSM, as well as provides soft function entities for model operations by combining with AGOS, while it provides domain application web services for end users, in order to accomplish system target. The functional structure of DSMEI is shown in Figure 7.1.

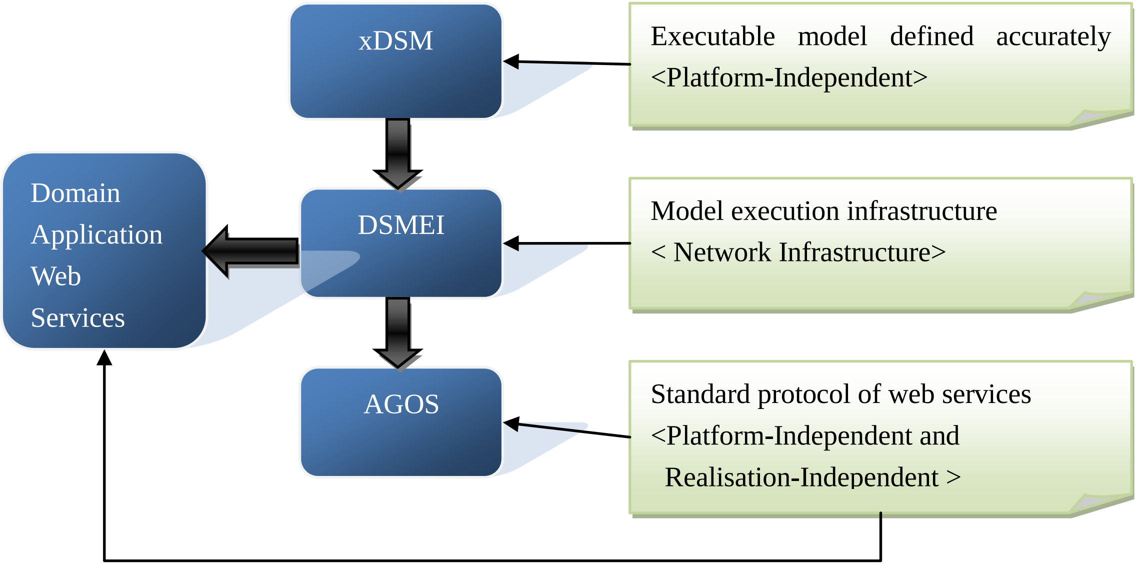

xDSM
Executable model defined accurately <Platform-Independent>
Domain Application Web Services
DSMEI
Model execution infrastructure < Network Infrastructure>
AGOS
Standard protocol of web services <Platform-Independent and Realisation-Independent >

Figure 7. 1 Functional Structure of DSMEI

DSMEI parses and executes xDSM which is a platform-independent executable model. Meanwhile, DSMEI employs web services as software functional entities which are platform-independent and realisation-independent based on the standard web service protocol system. On one hand, it provides atomic software functions for domain-specific system to express the utmost of the reusability and openness of web services; on the other hand, the customised domain application software functions corresponding to xDSM application model can be used by clients widely and standardisedly. DSMEI is open and substitutable since the relevant parts of DSMEI are platform-independent and realisation-independent. For example, different DSMEIs developed on different operating systems can provide the same execution environment for xDSM like model virtual machines.

The architecture of DSMEI is made up of BLEF (Behaviour Logic Execution Framework), DSPROF (PROvider Framework of Domain application web Services), AGOSOF (suppOrt Framework of AGOS), as shown in Figure 7.2.

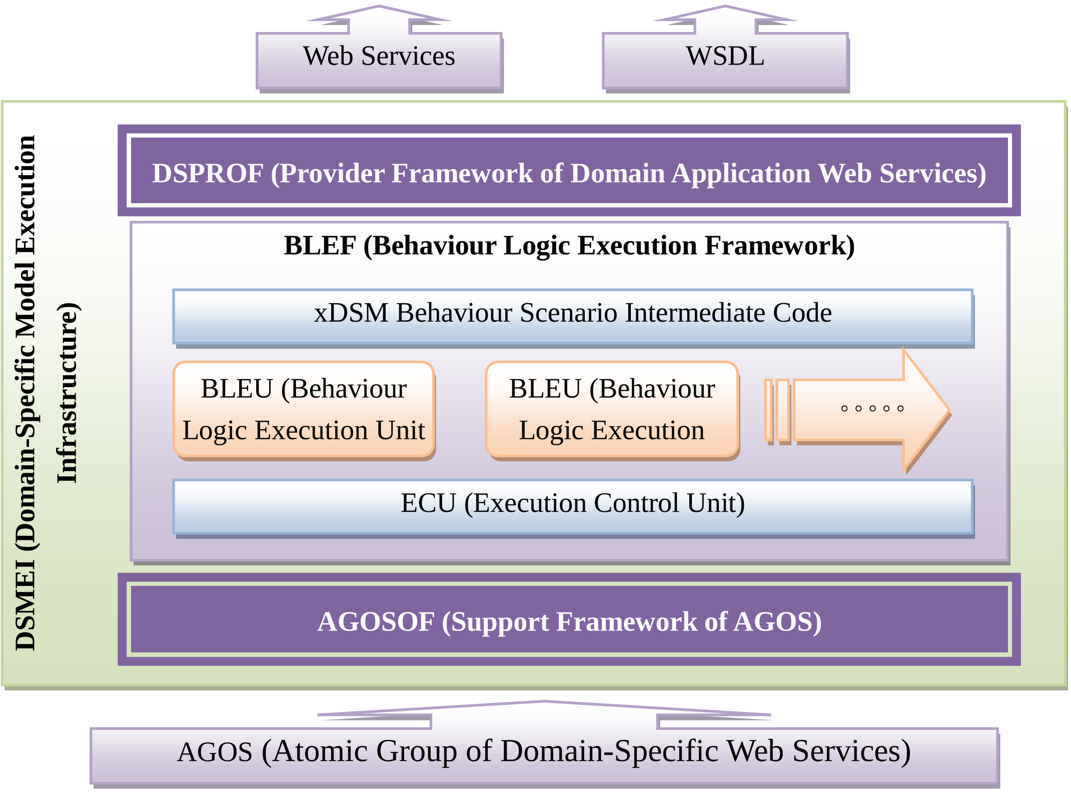

Web Services
WSDL
DSMEI (Domain-Specific Model Execution Infrastructure)
DSPROF (Provider Framework of Domain Application Web Services)
BLEF (Behaviour Logic Execution Framework)
xDSM Behaviour Scenario Intermediate Code
BLEU (Behaviour Logic Execution Unit
BLEU (Behaviour Logic Execution
○ ○ ○ ○ ○
ECU (Execution Control Unit)
AGOSOF (Support Framework of AGOS)
AGOS (Atomic Group of Domain-Specific Web Services)

Figure 7. 2 DSMEI Architecture

- **BLEF (Behaviour Logic Execution Framework)**

BLEF is the core of DSMEI. It is responsible for parsing and executing xDSM application model. Under the harmony and control of ECU (Execution Control Unit), BLEF creates BLEUs (Behaviour Logic Execution Unit) which is amount configurable to load and execute xDSM behaviour scenario intermediate code concurrently. BLEU is similar to a behaviour logic processor. BLEF involves:

- xDSM behaviour scenario intermediate code：xDSM BS is the basic behaviour logic unit of xDSM. BS is extracted from xDSM defined with XDML and compiled into intermediate code which is able to be parsed and executed by BLEUs.

- BLEU (Behaviour Logic Execution Unit): It is a software object. BLEU can parse and execute xDSM behaviour scenario intermediate code autonomously, control the execution state independently, handle the control flow and the data flow of operations execution, as well as interrupt the execution and receive the external information.

- ECU (Execution Control Unit): It manages and coordinates BLEU instances, as well as provides the uniform façade for the interaction between DSPROF, AGOSOF and the BLEU cluster. BLEU running is driven by ECU with message.

- **DSPROF (PROvider Framework of Domain application web Services)**

DSPROF provides software functions of xDSM application model for end users, which are BSs marked with deployed state in xDSM application model. DSPROF assorts starting point, end point, return point and message receiving of a BS with its pin sequence to build a web service dynamically, and provides corresponding WSDL for each web service. Consequently, it provides end users with xDSM model execution application interfaces according with the open standard.

- **AGOSOF (suppOrt Framework of AGOS)**

AGOSOF adopts web service model based on business document exchange as web service calling mechanism. It forms a dynamic flexible web services support framework depending on abstract operations and virtual services to replace the changeable concrete implementation details by web service virtualisation according to AGOS service configuration.

7.2 Behaviour Logic Execution Framework

BLEF (Behaviour Logic Execution Framework) is the main body of the execution of xDSM application model. xDSM application model cannot be executed directly. It is needed that to extract xDSM BS and compile the BS described by XDML originally into intermediate code. The intermediate code is loaded, parsed and executed by BLEU. Meanwhile, ECU is responsible for cooperating and managing the BLEU cluster, as well as providing the uniform façade for BLEF interacting with the external framework.

BLEU is a relatively independent component for executing models. ECU provides global environment, message bus and controls for executing and cooperating BLEUs. BLEU can load, clear, suspend and recovery xDSM behaviour scenario intermediate code and behaviour execution context dynamically. Meanwhile, several BLEUs can constitute the BLEU cluster which communicate and cooperate with each other by messages in order to reduce coupling. The structure is quite suitable for the distributed execution environment since the BLEU cluster can be distributed at different servers and managed uniformly by ECU. Furthermore, the amount of the BLEU instances at a certain server can be fixed according to the server performance so as to enhance the performance of model execution by expanding the hardware capacity and improve the concurrency.

7.2.1 xDSM Behaviour Scenario Intermediate Code

xDSM models described by XDML language cannot be executed directly by BLEF. It involves much structure information and visualisation information of model elements, and the primary meta-model of behaviour scenario also implies some behaviour semantics. BLEF can extract BS of xDSM models then compile and parse it into the intermediate code which contain the pure behaviour logic procedure and interface information of BS and can be loaded and executed by BLEU directly.

Behaviour semantics represented by BS is contained in the structural tokens collections of xDSM behaviour scenario intermediate code. It can be parsed as the corresponding operational semantics sequence [102, 101]. Behaviour scenario is understood as a series of operation steps which represent the signification of this behaviour scenario. BLEU parses xDSM behaviour scenario intermediate code and executes the corresponding operation according to operational semantics in tokens so as to achieve the corresponding system objectives of behaviour scenarios. xDSM behaviour scenario intermediate code is also defined by XML Schema based on XML meta-language as follows:

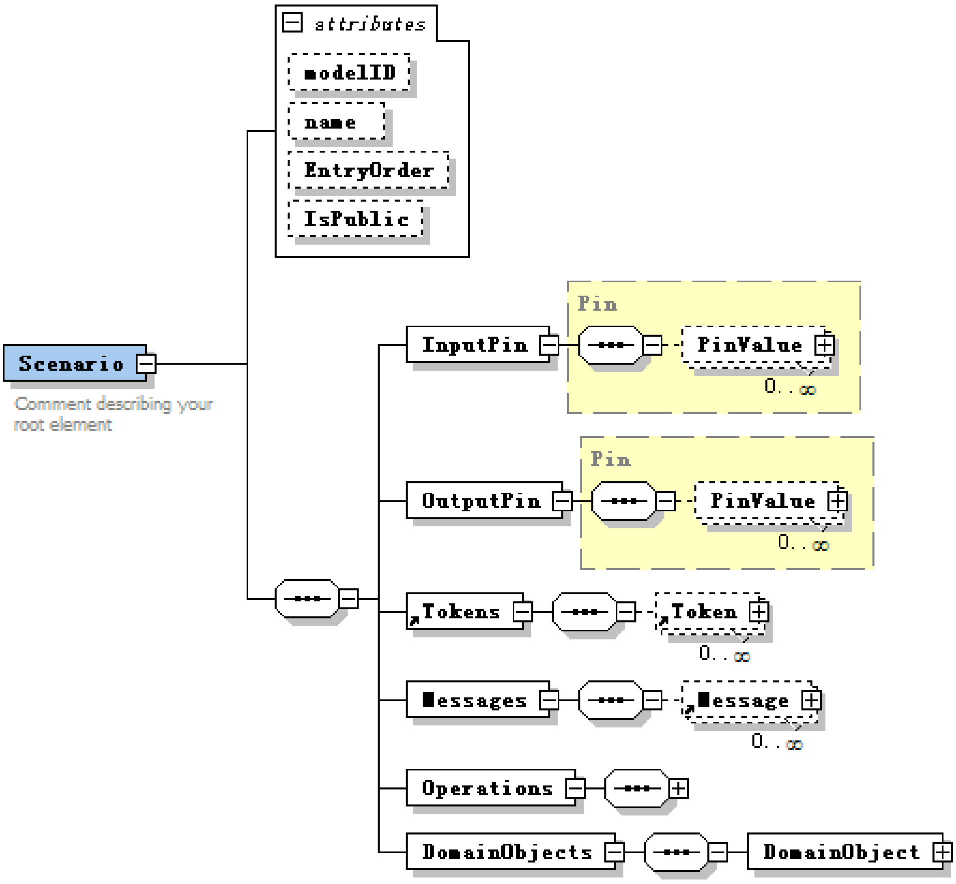


Figure 7. 3 XML Schema Definition of xDSM Behaviour Scenario Intermediate Code

Scenario is a root element of XML Schema. It represents the specific behaviour scenario. It involves:

Attributes:

- ModelID: It is the unique identification of a domain model, namely, the namespace of a BS.

- Name: It is the unique identification of a BS.

- EntryOrder: It is the serial number of the entry to execute tokens.

- IsPublic: It is the identification whether the BS is public or not.

Child Elements:

- InputPin: It is the input pin of a BS.

- OutputPin: It is the output pin of a BS.

- Tokens: It is the set of behaviour logic tokens of a BS. A token represents the minimal markup notation unit. It can express the concrete operation or data by the individual or grouped way. BLEU parses tokens to get data and executes specific operations.

- Messages: It is the set of messages received by BS. BS holds the execution state and waits for messages while a suspension takes place. These messages inputted with data will recover the execution process. MessageID is the unique identification of a message. EntryOrder is the serial number of the entry to execute tokens.

- Operations: It is the set of declarations of operations applied in a BS. Any BS operation in executing time should be searched and gotten from the set of Operations, and the different execution mode should be adopted according to the different concrete operation type. ModelID identifies the namespace of an operation. OperationName is the unique identification of an operation. Type involves Abstract Operations (corresponding to web services), Coordination Operations (corresponding to behaviour scenario), and Execution Framework API (corresponding to default internal functions).

- DomainObjects: It is the set of declarations of domain objects applied in a BS. DomainObject is a complex data type. It is the foundation of memory allocation and attributes access to a domain object in BLEU. ModelID identifies the namespace of DomainObject. Name is the unique identification of DomainObject.

7.2.2 xDSM Compiling and Parsing Algorithm

xDSM is translated into behaviour scenario intermediate code with a series of compiling and parsing processes in BLEF. Meanwhile, it involves three main processes named parsing AS&MC, parsing XDML and compiling xDSM BS. BLEF extracts and compiles xDSM BS to generate the intermediate code by the core algorithm in each process, and registers BS information and abstract operation information in DSMEI to lay the foundation for xDSM execution. The process is shown in Figure 7.4.

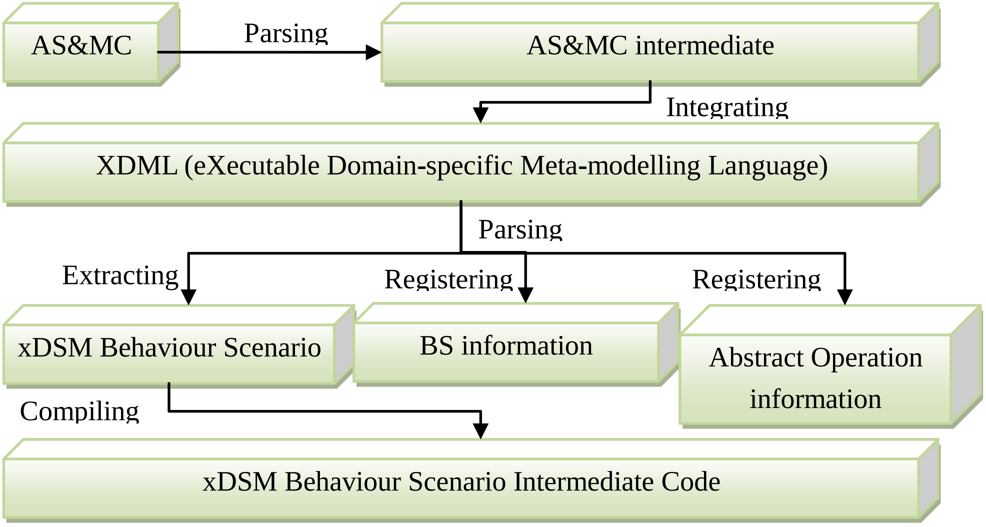

AS&MC
Parsing
AS&MC intermediate
Integrating
XDML (eXecutable Domain-specific Meta-modelling Language)
Parsing
Extracting
Registering
Registering
xDSM Behaviour Scenario
BS information
Abstract Operation information
Compiling
xDSM Behaviour Scenario Intermediate Code

Figure 7. 4 The Process of Compiling and Parsing xDSM BS Intermediate Code

1. AS&MC parsing algorithm

AS&MC parsing process is to parse action specifications and model constraints defined by AS&MC syntax into AS&MC intermediate format. The concrete syntax of AS&MC described by EBNF grammar is used to define action specifications and model constraints. Its representation is like advanced language.

Thereafter, AS&MC intermediate format is integrated into the concrete syntax of XDML which is based on XML meta-language. AS&MC intermediate format is a set of tokens which structure is as same as that of Tokens of xDSM behaviour scenario intermediate code. It is parsed and formed after modelling completed.

The thesis describes AS&MC parsing algorithm according to principles of compiler including lexical analysing algorithm, syntax analysing algorithm and semantics analysing algorithm, and constructs the finite automata by AS&MC EBNF definition to realise the lexical analyser. The input is the code of action specifications and model constraints defined by AS&MC syntax. AS&MC intermediate format is the output after parsed. The body of the algorithm is as follows:

- Firstly, Tokens and the category of each token are decomposed by AS&MC lexical analyser.

- Secondly, Tokens is traversed and done syntax analysis according to the EBNF definition of AS&MC so as to ensure Tokens is valid in syntax

- Afterwards, Tokens is traversed time after time and checked whether it meets the requirements of semantics according to the semantic rule set (for example: the variable must and only be used after it is declared). Finally, AS&MC intermediate format in line with the requirements of syntax and semantics is generated.

2. XDML parsing algorithm

The XDML parsing process is to extract behaviour logic of xDSM BS as well as provide necessary registration information (BS information, Abstract Operation information) for BLEF in order to connect with DSPROF and AGOSOF to support operation execution. XDML parsing algorithm includes main-procedure of

ParseXDML and sub-procedure of RegisterOperation.

- Main-procedure of ParseXDML：

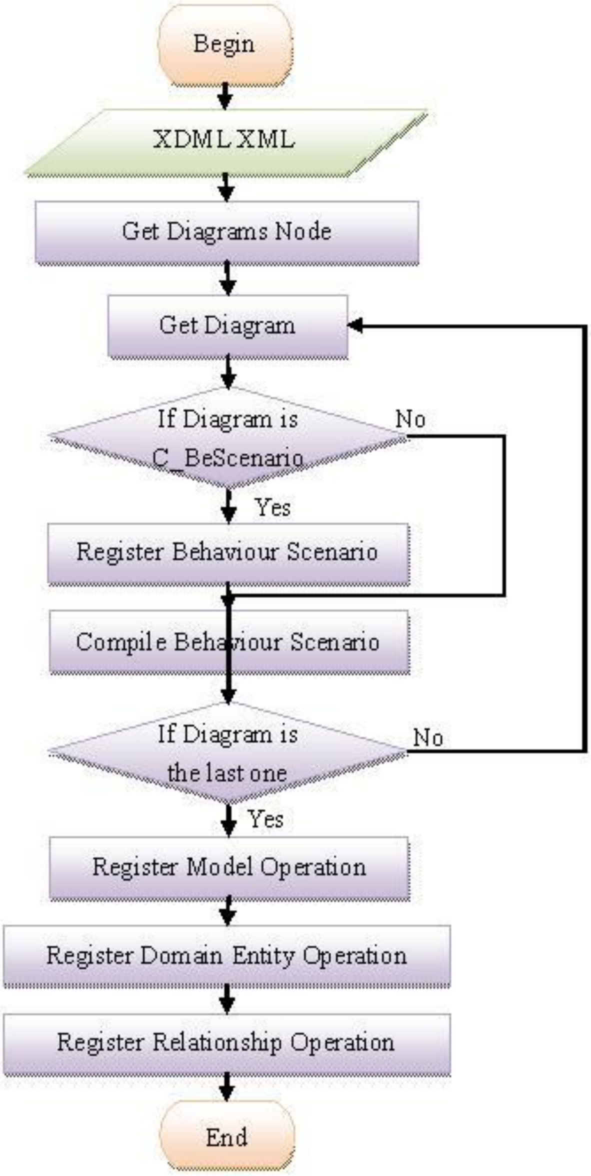

Begin
XDML XML
Get Diagrams Node
Get Diagram
If Diagram is C_BeScenario
No
Yes
Register Behaviour Scenario
Compile Behaviour Scenario
If Diagram is the last one
No
Yes
Register Model Operation
Register Domain Entity Operation
Register Relationship Operation
End

Figure 7. 5 The Algorithm Flow Chart of ParseXDML

```
Procedure ParseXDML (xDSM: TXMLNode)    // Main-procedure of parsing XDML
    Begin
        xDSM.GetNode('Diagrams');                    // Diagrams node is gotten
        Foreach(Diagram in Diagrams) Begin
            If (Diagram.type=C_BeScenario) then Begin
                RegisterBeScenario(Diagram);      // Behaviour Scenario is registered
                CompileBeScenario(Diagram);       // Behaviour Scenario is compiled
            End;
        End;
        RegisterOperation(Model. Operations)      // Model operation is registered
        xDSM.GetNode('Entities');                 // to analyse domain entity operation
        Foreach(Entity in Entities)
            RegisterOperation(Entity. Operations);
        Model.GetNode('Relationships');           // to analyse relationship operation
        Foreach(Relation in Relationships)
            RegisterOperation(Relation. Operations);
    End;
```

- Sub-procedure of RegisterOperation：

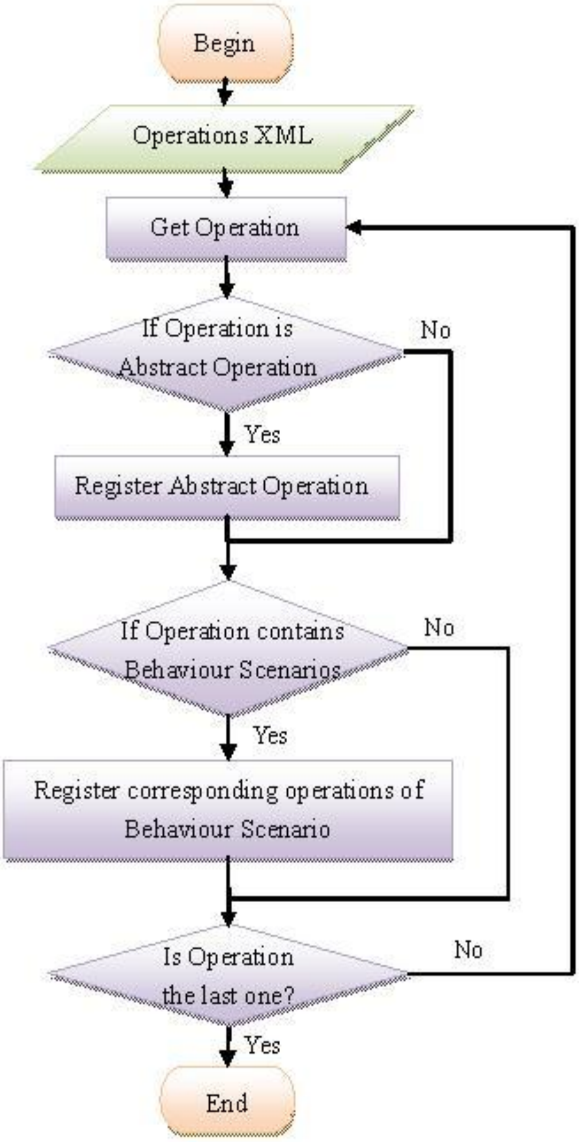

Begin
Operations XML
Get Operation
If Operation is Abstract Operation
No
Yes
Register Abstract Operation
If Operation contains Behaviour Scenarios
No
Yes
Register corresponding operations of Behaviour Scenario
Is Operation the last one?
No
Yes
End

Figure 7. 6 The Algorithm Flow Chart of RegisterOperation

```
Procedure RegisterOperation(Operations: TXMLNode) //Sub-procedure of registering operation
    Begin
        Foreach(Operation in Operations) Begin
            If (Operation.IsAbstract) then
                RegisterAbOperation(Operation);        // Abstract Operation is registered
            Else if(Operation.BSID<>null) then
                RegisterOPtoBS(Operation, BSID); // the related operations of Behaviour
```

```
                                                    Scenario are registered
      End;
   End;
```

The XDML parsing process is to traverse the model defined by XDML, extract BS from Diagrams and register including its namespace (ModelID) and BS ID. Meanwhile, the behaviour scenario is compiled and the information received by the behaviour scenario is registered. All operations contained in Model, Entity and Relationship are traversed one by one. Abstract Operation is extracted and registered including its namespace (ModelID), operation name and Pin. Moreover, coordination operations corresponding to the behaviour scenario are added and registered, including its Pin, IsPublished or not.

3. xDSM behaviour scenario compiling algorithm

The xDSM BS compiling process is to extract behaviour semantics of model elements from the behaviour scenario as well as integrate and compile action specifications and running time constraints into xDSM behaviour scenario intermediate code so as to BLEU can understand and execute the corresponding xDSM behaviour scenario. xDSM behaviour scenario compiling algorithm is shown as follows：

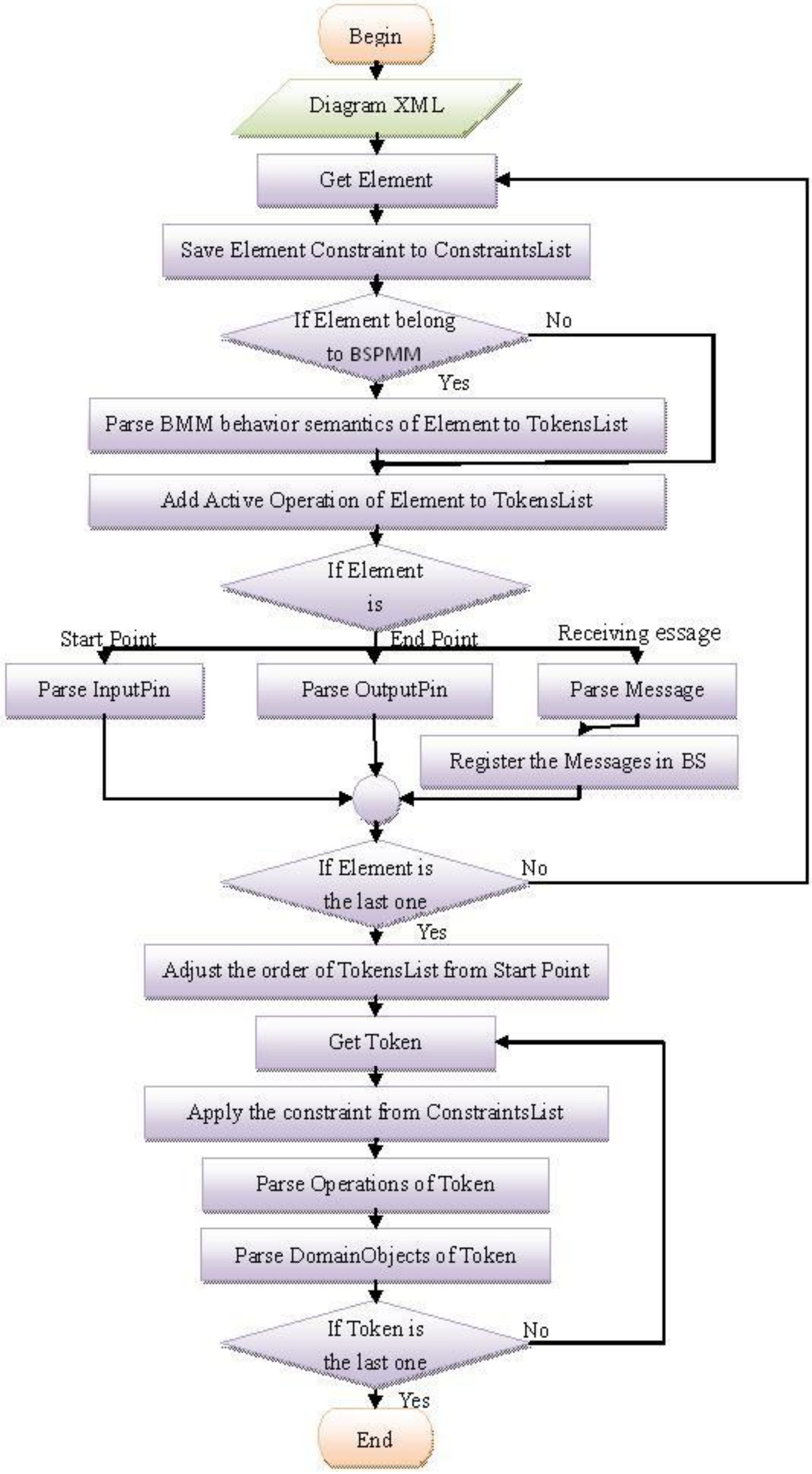

Begin
Diagram XML
Get Element
Save Element Constraint to ConstraintsList
If Element belong to BSPMM
No
Yes
Parse BMM behavior semantics of Element to TokensList
Add Active Operation of Element to TokensList
If Element is
Start Point
End Point
Receiving essage
Parse InputPin
Parse OutputPin
Parse Message
Register the Messages in BS
If Element is the last one
No
Yes
Adjust the order of TokensList from Start Point
Get Token
Apply the constraint from ConstraintsList
Parse Operations of Token
Parse DomainObjects of Token
If Token is the last one
No
Yes
End

Figure 7. 7 The Algorithm Flow Chart of CompileBeScenario

```
Procedure CompileBeScenario(Diagram: TXMLNode);
Begin
    // to traverse elements of Behaviour Scenario
    Foreach(Element in Diagram) Begin
        // to save element constraints into ConstraintsList
        ConstraintsList.Add(Element, Element.Constraints);
//if the element is the element of the primary meta-model of behaviour scenario, its behaviour semantics is parsed into TokensList
        If (Element.Type is BMM) then
            ParseBMMToTokensList(Element);
        //to add active operations and constraints of the element
        Element.GetNode('Operations');
        Foreach(Operation in Operations) Begin
            If (Operation.IsActive) then Begin
                ParseOPToTokensList(Operation);
                ApplyConstraints(Operation.Constraints)
            End;
        End;
        //if the element is the starting point, it is parsed into InputPin and marked as Starting Point
        If (Element.Type=C_Start) then Begin
            ParseInputPin(Element);
            StartElement := Element;
        End;
        //if the model element is the end point, it is parsed into OutputPin
        If (Element.Type=C_End) then
            ParseOutputPin(Element);
        //if the element is Message Receiving, it is parsed into Messages and the message receiving information of behaviour scenario is registered
```

```
            If (Element.Type=C_ReciveMessage) then Begin
                ParseMessages (Element);
                RegisterMGtoBS(Element, BSID) ;
            End;
        End;
        //to djust the order of TokensList from Starting Point according to the control flow
        AdjustTokensListOrder(TokensList, StartElement);
        //to traverse TokensList, add element constraints and extract Operations and DomainObjects used by Tokens
        Foreach(Tokens in TokenList) Begin
            ConstraintsList .ApplyConstraints(Tokens);
            ParseToOperations(Tokens);
            ParseToDomainObjects(Tokens);
        End;
        // to save TokensList into Tokens
        TokenList.SaveToTokens;
    End;
```

The compiling process of xDSM BS is to traverse elements and their affiliated operations and constraints firstly, and save the related tokens of each element into TokensList. For each element:

- To save element constraints into ConstraintsList. This is the preparation for adding constraints into the related tokens.

- If the element is the element of the primary meta-model of behaviour scenario, its behaviour semantics is parsed into TokensList. For example, the element of Judgement Entity is parsed as the tokens of "*if (Expression) then ... else ....*"

- To add active operations and constraints of the element. The constraints are divided into pre-conditions and post-conditions, which are added before or after operation tokens directly as AS&MC intermediate format.
- If the element is the starting point, it is parsed into InputPin and marked as Starting Point.
- If the element is the end point, it is parsed into OutputPin.
- If the element is Message Receiving, it is parsed into Messages and the message receiving information of behaviour scenario is registered

Afterwards to adjust the order of tokens in the TokensList from Starting Point according to the control flow, to traverse TokensList then add element constraints and extract Operations and DomainObjects used by Tokens, and to save TokensList into Tokens, which make xDSM behaviour scenario intermediate code come into being finally.

7.2.3 Behaviour Logic Execution Unit

BLEU (Behaviour Logic Execution Unit) is the relatively independent component for model execution. It interprets and executes xDSM behaviour scenario intermediate code within BLEF autonomously, and realises xDSM behaviour logic to accomplish the given system objective. Behaviour scenario is the behaviour logic unit of xDSM, which is transformed into xDSM behaviour scenario intermediate code by the parsing and compiling processes. BLEU loads xDSM behaviour scenario intermediate code, and executes the intermediate code by the way of interpretive execution. Meanwhile, it creates processes and memory spaces independently and manages the control flow and the data flow of behaviour logic execution by itself. The logic structure of BLEU is as follows:

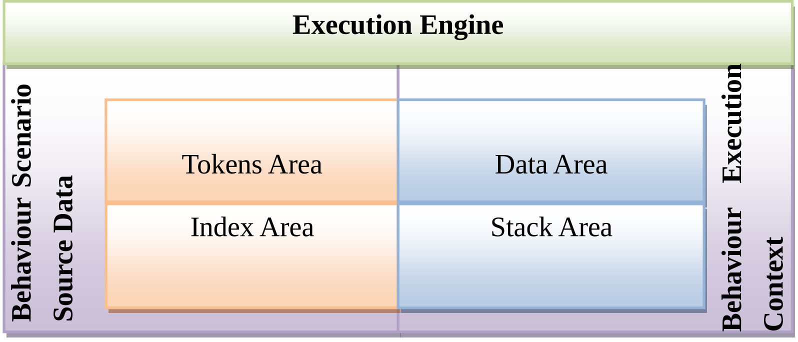


Figure 7. 8 The Logic Structure of BLEU

Data in tokens area and index area is determined by xDSM behaviour scenario intermediate code. It is relatively fixed and composes meta-data of BSs. Data area and stack area may be different in different execution instances. They compose behaviour execution context.

- Tokens Area: It is the memory area of code tokens and responsible for loading Tokens of xDSM behaviour scenario intermediate code. It is the concrete expression of behaviour logic.

- Index Area: It is responsible for loading data of BS (attributes, InputPin, OutputPin), lists of Messages, Operations and DomainObjects of each segment of xDSM behaviour scenario intermediate code in sections. Index area is the memory area of meta-data of BS, which provides essential information for message receiving, operations searching and executing, and memory assignment and access of complex objects.

- Data Area: It is the memory area of data during BLEU running process. It is the self-managed memory spaces. Variables, constants and domain object instances are stored in this area.

- Stack Area: It is the memory area of the related data of operations (such as parameters, return values etc.) and execution context (such as the execution token pointer etc.) during BLEU running process. Stack area is the core area of operations execution and the necessary condition for executing operations.

Execution engine is the core of BLEU, which drives and executes the behaviour logic of xDSM BS and realises the operation semantics of xDSM behaviour scenario intermediate code by the way of interpretative execution. The algorithm of execution engine running is as follows:

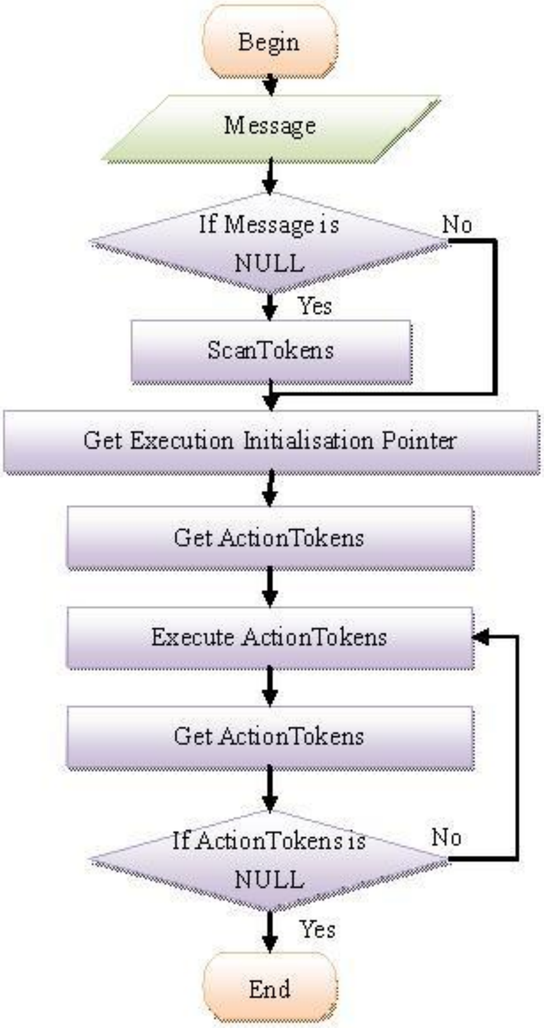


Figure 7. 9 The Algorithm Flow Chart of XEngineRun

```
Procedure XEngineRun (Message: TMessage)
Begin
    If Message=null then
        PrescanTokens(Tokens);                        //to scan Tokens
    ActivityToken := GetStartTokenOrder(Message);          // to get ActivityToken
    ActionTokens := GetActionTokens(ActivityToken, Tokens);   //to get Action Tokens
    While (ActionTokens<>null) Begin
        ActivityToken := DoAction(ActionTokens);           //to execute Action Tokens
        ActionTokens := GetActionTokens(ActivityToken, Tokens);
    End;
End;
```

BLUERun starts from Starting Point and terminates at End Point or suspends execution. If it suspends execution, the execution can also be continued by Message Receiving. Therefore, both Starting Point and Message Receiving can start BLUERun. Message should be null if it starts from Starting Point. The running steps of BLEU are as follows:

- To scan Tokens to initialise execution context when behaviour logic execution starts from Starting Point, as well as to assign memory for variables and domain objects.

- To get the execution token pointer -- ActivityToken in tokens area. The serial number of Token is in the attributes of EntryOrder of xDSM behaviour scenario intermediate code if the execution starts from Starting Point; the serial number of Token should be in the attributes of EntryOrder of the corresponding Message if the execution starts from Message Receiving.

- To sequential down and analyse tokens from the one in tokens area pointed by ActivityToken to get ActionTokens which represents an Action (for examples, expression evaluation action, group action, etc.).

- Loop until ActionTokens are null.

- To execute the action represented by ActionTokens. It is accomplished by the concrete action execution function according to the action adscription. The execution process of the action may be recursive, such as complex expression evaluation action. After accomplishing the action execution, subsequent ActivityToken can be gotten in term of behaviour logic.

- To sequential down and analyse tokens from the one in tokens area pointed by ActivityToken to get ActionTokens

It is very suitable for using BLEU to execute the consecutive BS which is not suspended and without Message Receiving. The domain-specific business logic can be realised better with the architecture. Meanwhile, BLEU provides the mechanism of suspension and message receiving. The execution state of BLEU can be reserved by suspension as well as the execution can be resumed by message receiving and the execution token pointer -- ActivityToken can be relocated. The mechanism can deal with the BS better that the single session is executed discontinuously. For instances, BS of asynchronous operations and the interface related BS. For the interface related BS, the client is required to support the serialisable interface updating techniques, such as Ajax techniques [36], which can update local pages in web browser via transferring HTML by web services.

7.2.4 Execution Control Unit

ECU (Execution Control Unit) is employed by BLEF to manage and harmonise the BLEU cluster. It is an instance of Façade Design Pattern, providing the uniform façade for the interaction between DSPROF, AGOSOF and the BLEU cluster.

Meanwhile, ECU connects with DSMEI frameworks by the way of message delivery which is coupled loosely, and drives the running of BLEU. The core functions of ECU are expressed as follows:

➢ To manage the BLEU cluster

ECU manages the execution cycle of BLEU. It monitors all current states (Execution, Idle and Interruption) of BLEU by the BLEU state list during the running process, and manages BSs running in a certain session of BLEU by the BLEU execution scenario list. The execution cycle of BLEU is illustrated as follows.

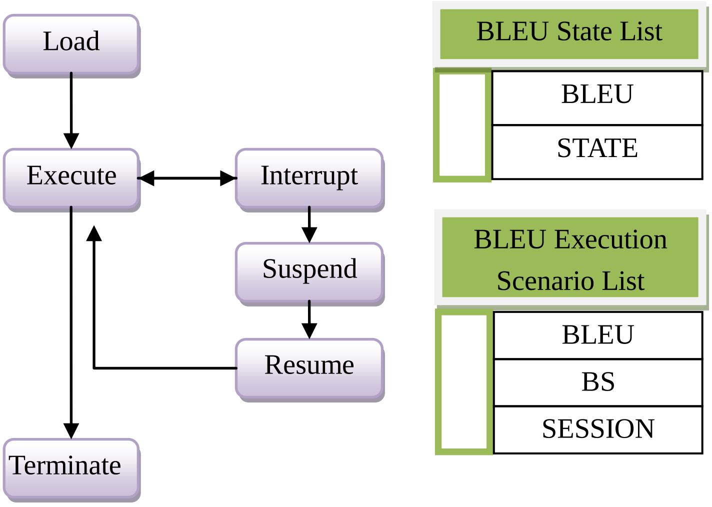


Figure 7. 10 The Execution Cycle of BLEU

- Load：When ECU receives the message of execution request, it queries the BLEU state list and assigns the execution task to idle BLEU. If there is no idle BLEU, on one hand, messages with execution task are queuing and waiting; on the other hand, the interrupted BLEU execution context is suspended to release available BLEU. Once BLEU is gotten, its state is changed to Execution and the corresponding xDSM behaviour scenario intermediate code is loaded and executed and the BLEU execution scenario list is registered with BLEU instance, executed Behaviour Scenario and the corresponding session.

- Execute: BLEU runs according to behaviour logic of BS.

- Interrupt: When BLEU execution meets interruption and waits for receiving messages, its state is changed to Interruption.

- Suspend: ECU serialises and saves BLEU execution context (behaviour execution context, execution token pointer, etc.) from the BLEU with Interruption state, terminates the execution of the BLEU and changes its state to Idle, at the same time, clears the information of the BLEU in the BLEU execution scenario list.

- Resume: After receiving the message that the BS in session interrupts waiting, ECU will assign the idle BLEU to load the corresponding xDSM behaviour scenario intermediate code, anti-serialise the BLEU execution context which is saved when suspended and change the BLEU information in the BLEU execution scenario list.

- Terminate: When BLEU execution is completed and returned, its state is changed to Idle, BS source data and behaviour execution context are cleared, and the information in the BLEU execution scenario list is deleted.

➢ **Uniform Façade**

ECU provides uniform façade for DSPROF. It receives the message of BS execution request and sends execution results or exceptions. DSPROF provides domain application web services for the external according to the registration information of BS in DSMEI. Web service call is transformed into BS execution request message and BLEU is dispatched by ECU to execute the corresponding BS.

When the execution terminates, execution results or exceptions will be returned by BLEU and sent to DSPROF by ECU.

ECU provides uniform façade for AGOSOF. It sends the abstract operation request and receives running results or exceptions returned by the support services. AGOSOF manages AGOS and provides software functional entities by the way of web services for Abstract Operation. ECU sends the message of executing abstract operation to AGOSOF when receiving the execution request of Abstract Operation from BLEU. AGOSOF calls the corresponding web services according to the registration information of Abstract Operation and returns execution results or exceptions. ECU receives the returned message and feeds back to BLEU.

ECU provides uniform façade for BLEU. It provides the related global control functions for BLEU by the way of interfaces, including cooperation operation call (xDSM Behaviour Scenario), abstract operation call (web services), message receiving and sending, and global data access.

- **Message delivery and control**

The three main cooperating frameworks of DSMEI are driven by messages. Messages are among web services and BLEU. In BLEU, all messages are related to tasks, and messages are the concrete expressions of tasks. ECU provides uniform façade for the three frameworks, delivers messages for tasks, and accomplishes the execution of BS and Abstract Operations call. During the process, ECU can control messages receiving and sending, filter messages and control the priority of messages or pre-handle messages.

7.3 Web Service Model based on Business Document Exchange

DSMEI is logically divided into the internal framework and the external framework. The internal framework is BLEF which is the core of DSMEI and driven by messages. The external framework is composed of DSPROF and AGOSOF. DSPROF provides domain application web services for the external by executing xDSM models. AGOSOF calls external web services as the computational logic of xDSM model execution. Web services run through the whole DSMEI, and drive the collaborative operation of the internal framework and the external frameworks by mutual mapping between DSMEI message specifications of the internal framework and SOAP protocol of the external framework. However, web services provided by DSMEI are changing with xDSM application model. They are virtual services. At the same time, AGOS used by DSMEI is unfixed and web services to realise abstract operations are also changing with requirements. The variability of DSMEI determines that web services cannot be bound or published permanently as well as cannot mapping SOAP and DSMEI messages permanently. Therefore, DSMEI needs a flexible web service application model.

Web services are based on XML and supported by SOAP. From the perspectives of business functionalities and data exchange, web services are software functional entities to realise WSDL and SOAP standard business document exchange. As a business document with business function identifier and business dataset, SOAP message can be generated and parsed dynamically. On one hand, SOAP communication is an exchanging process of SOAP service calling request message and SOAP service result return message. It is an exchanging procedure of SOAP standard business documents. SOAP communication procedure can be accomplished by dynamically parsing and generating SOAP business documents, which replaces the traditional web services binding and communication procedure. Meanwhile, web services publishing can be regarded as WSDL document exchange procedure. On the other hand, through business documents mapping, SOAP and WSDL documents can be transformed into DSMEI messages dynamically, and DSMEI messages can also be transformed into SOAP and WSDL documents. So the internal framework and the external frameworks of DSMEI are joined. Based on the above discussion, the thesis presents web service model based on business document exchange as shown in Figure 7.11.

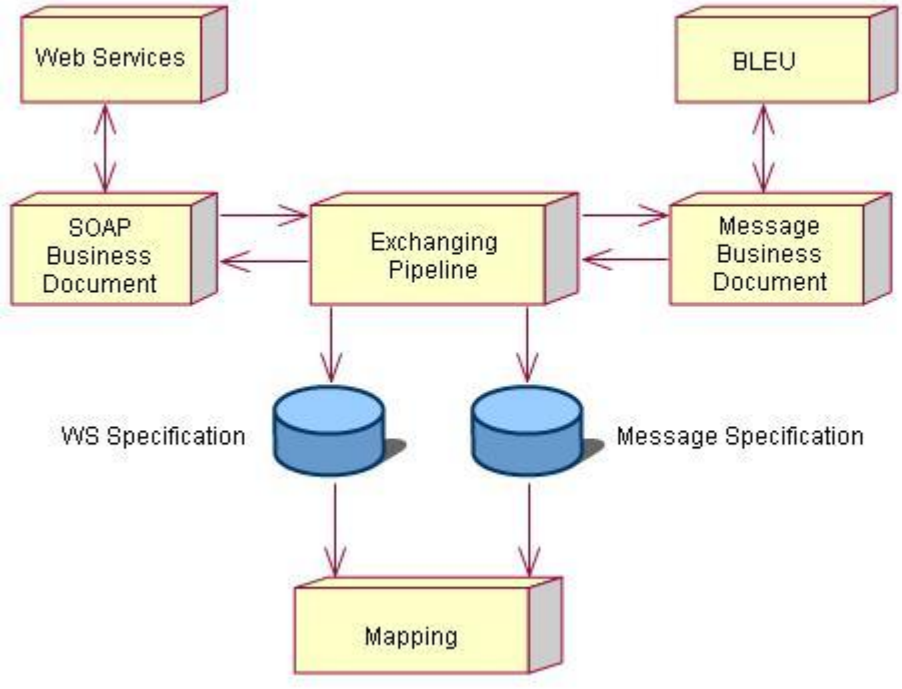


Figure 7. 11 Web Service Model based on Business Document Exchange

Web service model based on business document exchange takes the traditional protocol and messages as the corresponding business documents (business function identifier and business dataset). Business documents are divided into two categories. One is SOAP business document generated by the standard web service, the other is message business document generated by BLEU. The exchange between the documents in the same category is a standard call which accords with the established protocol and specification, such as web service call. The exchange between the documents in the different categories is accomplished by Exchanging Pipeline transforming and transferring business documents. The core of Exchanging Pipeline is business documents mapping that Message business documents and SOAP business documents are transformed according to web service specifications and message specifications.

- Web Services and SOAP Business Documents: web service is a standard software entity which follows SOAP protocol. The related SOAP message is the business document containing business function identifier and business dataset. Web service call can be regarded as an exchanging procedure of SOAP business documents.

- BLEU and Message Business Documents: BLEU is the main body to carry out domain application web services and the entity to call AGOS service. It is driven by messages. Messages in DSMEI are also business documents containing business function identifier and business dataset. Abstract operation execution and xDSM BS execution are driven by Message business documents. They are transformed into standard web service calling procedures by exchanging Message business documents and SOAP business documents.

- Exchanging Pipeline: Exchanging Pipeline realises the exchange between SOAP business documents and Message business documents. The core of Exchanging Pipeline is business document mapping. Meanwhile, it provides an Exchanging Pipeline to deal with business documents. It processes business documents step by step, for instances, business document filtering, logging and security controlling. Exchanging Pipeline is the core transforming and transferring entity of business documents.

- Web service specification and message specification: They are the structures of business documents and validity rules. Web service specification defines the specification of SOAP business documents and message specification defines the specification of Message business documents.

- Mapping: Mapping is the main function of Exchanging Pipeline. It establishes the mapping criterion with source specifications and target specifications, and transforms the business document with source specifications into that with target specifications. It is the exchange between SOAP business documents and Message business documents.

There are two main processes involved in web service model based on business document exchange in DSMEI. They are SOAP business document exchanging process of web service call, and the exchanging process of SOAP business documents and Message business documents.

- SOAP business document exchanging process of web service call: It is a standard web service call process: document transformation is accomplished by the concrete service execution entity, and web service call procedure is accomplished by the exchange of SOAP service calling request document and SOAP service result return document. Two procedures are involved in the process:

  - Calling procedure of domain application web services: SOAP service calling request document generated by External Call is transferred to DSMEI and SOAP service result return document is generated by executing BS in DSMEI to accomplish web service call.

  - AGOS service calling procedure: DSMEI generates SOAP service calling request document and gets SOAP service result return document by executing the external web services to accomplish actual AGOS service call.

- The exchanging process of SOAP business documents and Message business documents: It is done by Exchanging Pipeline. It makes SOAP business documents and Message business documents transforming into each other and loosely couples the internal framework and the external framework of DSMEI to realise the exchange of business request and data. There are four operations contained in the process as follows:

  - In the calling procedure of domain application web services, SOAP service calling request document is transformed into BS execution request message document.

  - In the calling procedure of domain application web services, BS execution result message document is transformed into SOAP service result return document.

  - In AGOS service calling procedure, abstract operation execution request message document is transformed into SOAP service calling request document.

  - In AGOS service calling procedure, SOAP service result return document is transformed into abstract operation execution return message document.

Web service model based on business document exchange joins the internal framework and the external frameworks of DSMEI effectively. It provides a flexible and dynamic web service calling and publishing mechanism which makes dynamic domain application web service calling and publishing be possible. Meanwhile, AGOS services is virtualisable by document mapping and SOAP business documents dynamically generating, which makes DSMEI with strong reactivity and evolutionary.

7.4 Provider Framework of Domain Application Web Services

DSPROF (PROvider Framework of Domain application web Services) is the external framework of DSMEI. It provides open and standard application interfaces of xDSM model execution for end users by web services. A series of BSs are established to system objectives in the process of xDSM modelling. Each BS has its Starting Point and End Point. Besides, some BSs also include Return Point and Message Receiving. These modelling elements all contain parameter Pin which compose multi-group of input parameters and output parameters of a BS. Therefore, each BS corresponds to one main operation and may attach several message interaction procedures. The main operation and message interaction procedures of the BS which is marked as *public* is released as domain application web services by DSPROF.

7.4.1 Structure of DSPROF

DSPROF adopts web service model based on business document exchange to construct the standard web service architecture. It connects BLEF by Exchanging Pipeline and carries out web services by executing xDSM models. DSPROF provides standard web services for the external by the way of SOAP business document exchange and provides the corresponding WSDL web services description at the same time. For the internal, DSPROF transforms SOAP business documents and Message business documents according to BS Registration Information. The structure of DSPROF is shown in Figure 7.12.

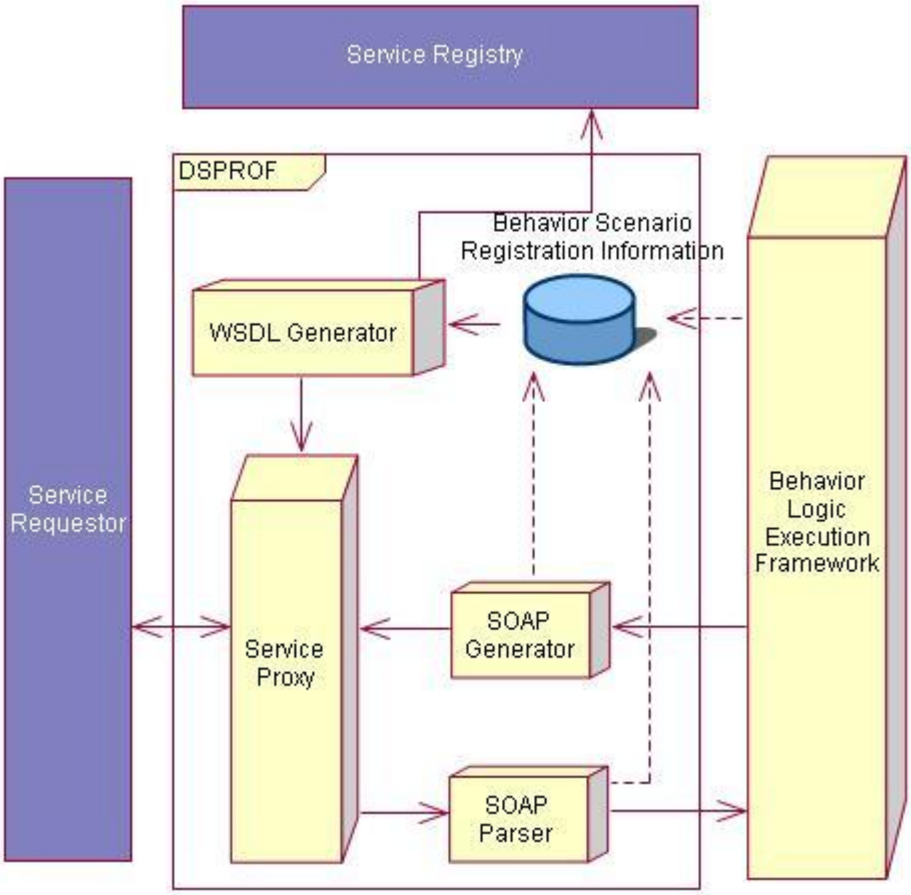


Figure 7. 12 The Structure of DSPROF

DSPROF is composed of BS Registration Information, Service Proxy, SOAP Parser, SOAP Generator and WSDL Generator.

- **Behaviour Scenario Registration Information**

DSMEI generates BS Registration Information in the process of parsing and compiling xDSM application model. It is web service specification and message specification of DSPROF. The main operation and each Message published in BS Registration Information will be transformed into a domain application web service. BS Registration Information is a structural dataset as follows.

Table 7. 1 BS Registration Information

ModelID		Identification of a xDSM model. It is the namespace of a BS
BSID		Unique identification of a BS
IsPublic		It is the identification whether BS is public or not.
MainOperation	Name	Name of the main operation represented by BS
	InputPin	InputPin of the main operation
	OutputPin	OutputPin of the main operation
MessageCount		The number of messages received by BS
Message0	MessageID	MessageID corresponding to Message0. It is the unique identification of Message0.
	InputPin	InputPin of Message0
	OutputPin	OutputPin of Message0
MessageN	…..	The same as Message0, described in parallel

- **Service Proxy**

Service Proxy is an interactive proxy between Service Requester and DSPROF. It provides SOAP protocol specification according to web service standard, receives web service requests (SOAP service calling request, WSDL request) and returns corresponding results (SOAP service result return, SOAP service error return, WSDL service description). It makes Service Requester invokes domain application web services by the transparent and standardised way.

- **SOAP Parser**

SOAP Parser is responsible for parsing SOAP service calling request documents received by Service Proxy in accordance with BS Registration Information, and transforming them into BS execution request message documents then transferring to BLEF.

- **SOAP Generator**

SOAP Generator is responsible for parsing BS execution result message documents returned by BLEF in accordance with BS Registration Information, and transforming them into SOAP service result return documents then transferring to Service Proxy.

- **WSDL Generator**

WSDL Generator generates the WSDL description documents of the web service which is transferred to Service Proxy in accordance with BS Registration Information.

7.4.2 Domain Application Web Service Call

For end users, domain application web service call is the standard web services realised by SOAP protocol. It is a SOAP business document exchange process within DSMEI which receives SOAP service calling request documents and returns SOAP service result return documents to accomplish domain application web service call. Meanwhile, SOAP business documents and Message business documents exchange need to be accomplished in order to realise web services by executing xDSM models. Domain application web service call process in DSPROF is as follows:

1. Service Proxy monitors HTTP requests and gets SOAP service calling request document from HTTP body. It holds session then sends SOAP service calling request document to SOAP Parser with the session ID.

2. SOAP Parser extracts the service method name and the parameter list from SOAP service calling request document, and searches for the suited

MainOperation.Name or Message.MessageID of the corresponding ModelID in BS Registration Information. The BS execution request message document is generated in terms of the corresponding BS items involving ModelID, BSID, SessionID, MainOperation.Name or Message.MessageID and the input parameter list, and sent to ECU within BLEF.

3. ECU assigns BLEU to handle the BS execution request message document, executes the related BS and gets the BS execution result message document which is sent to SOAP Generator by ECU.

4. SOAP Generator parses the BS execution result message document and gets ModelID, BSID, and SessionID etc. If the execution result is an exception, SOAP service fault return document will be generated and the fault information will be filled in FaultCode and FaultString; or else SOAP service result return document will be generated in terms of the suited BS items searched in BS Registration Information and the returning value will be filled in Return element. Then SOAP service return document and the session ID will be sent to Service Proxy.

5. Service Proxy returns SOAP service return document as the result of web service call to Service Requestor according to the session ID to accomplish domain application web service call.

In the process of domain application web service call, SOAP business documents are exchanged by Service Proxy and web service call is openly and standardisedly done by Service Requestor. Meanwhile, Exchanging Pipeline that is composed of SOAP Parser and SOAP Generator accomplishes the mapping between SOAP business documents and Message documents with BS Registration Information used as web service specification and message specification. So domain application web services are realised by xDSM model execution.

7.4.3 Web Service WSDL Description Generation

WSDL is a contract language based on XML which is used to describe web services, its parameters and return values. Firstly, WSDL describes the accessing operations and the request/response messages abstractly. Then WSDL binds these to the concrete transfer protocol and the message format. WSDL descriptions of web services can be modelised as two parts: to describe the service interface in the part of abstract definition, and to describe the service implementation in the part of concrete definition:

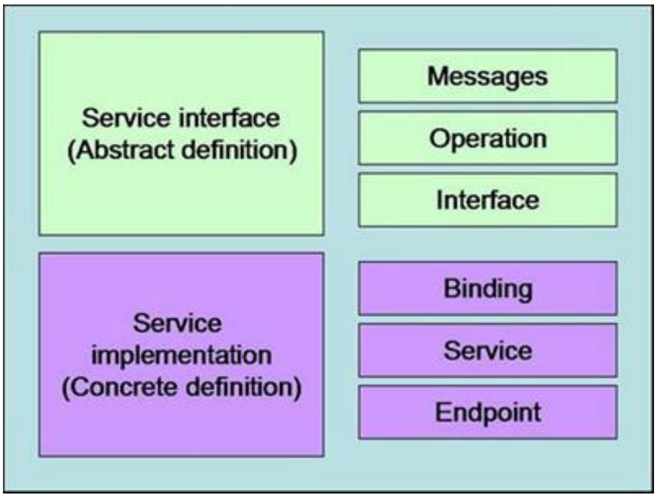


Figure 7. 13 WSDL Concept Component Model

WSDL binding defines the message format and SOAP protocol details for web services in order to use them directly. UDDI also needs web services description with WSDL. It is the key to using domain application web services. WSDL descriptions of domain application web services are accordant with SOAP business document exchange of domain application web services call.

DSPROF gets the request for web services WSDL descriptions from Service Proxy and sent it to WSDL Generator. WSDL Generator gets ModelID from the request message, generates WSDL descriptions according to BS Registration Information for the specific xDSM model, and sends it to Service Proxy for return. WSDL description generation is the core of the above process. It builds the parts of abstract definition and concrete definition of WSDL description according to BS Registration Information.

- Message: it is used to define data in communication. Input Pins and Outpun Pins of MainOperation and all Messages in BS are used to build Messages of WSDL.

- Operation: it associates the message exchange pattern with one or more messages. InputPins and OutputPins of MainOperation or Messages in BS are organised and mapped to the messages of WSDL to form Operations.

- Interface: Operations are polymerised based on transfer and message neutrality. They are organised by MainOperation and Messages in Behaviour Scenario to form the Interface.

- Binding: it specifies the transfer mode and message format of Interface.

- Endpoint: it associates URL (Uniform Resource Locator) with Binding. It defines the sub-element *location* of <Soap:address> as the URL address of this service based on WSDL Binding.

- Service: Endpoints of BS interfaces are aggregated to form service.

7.5 Support Framework of AGOS

AGOSOF (suppOrt Framework of AGOS) is another external framework of DSMEI. It adopts web services as software entities for xDSM model execution. A series of abstract operations are defined during xDSM modelling in order to encapsulate the specific computational logic and implement the refined software functions. DSMEI adopts AGOS to implement the corresponding abstract operations in xDSM models. Meanwhile, a dynamic flexible web services support framework is formed by virtualising web services according to AGOS service configuration. It employs abstract operations and virtual services to substitute the easy-changed implementation details.

7.5.1 Structure of AGOSOF

Service-oriented development beyond the traditional application development methods fully considers about the existing service and dynamically deploys web service resource during software life circle, including services of querying, matching, assembling, and replacing. So AGOSOF should support not only SOAP protocol which is web service foundation, but also the technologies and methods of service assembling and replacing. AGOSOF adopts web service model based on business document exchange to utilise AGOS based on business document exchange.

The execution of web services is driven by xDSM model execution. Web service virtualisation is implemented by Exchanging Pipeline in the exchange process between message business document and SOAP business document for corresponding to the implementation of abstract operations. AGOSOF calls the standard web services by the way of SOAP business document exchange for the external to implement SOAP protocol. The structure of AGOSOF is shown as follows:

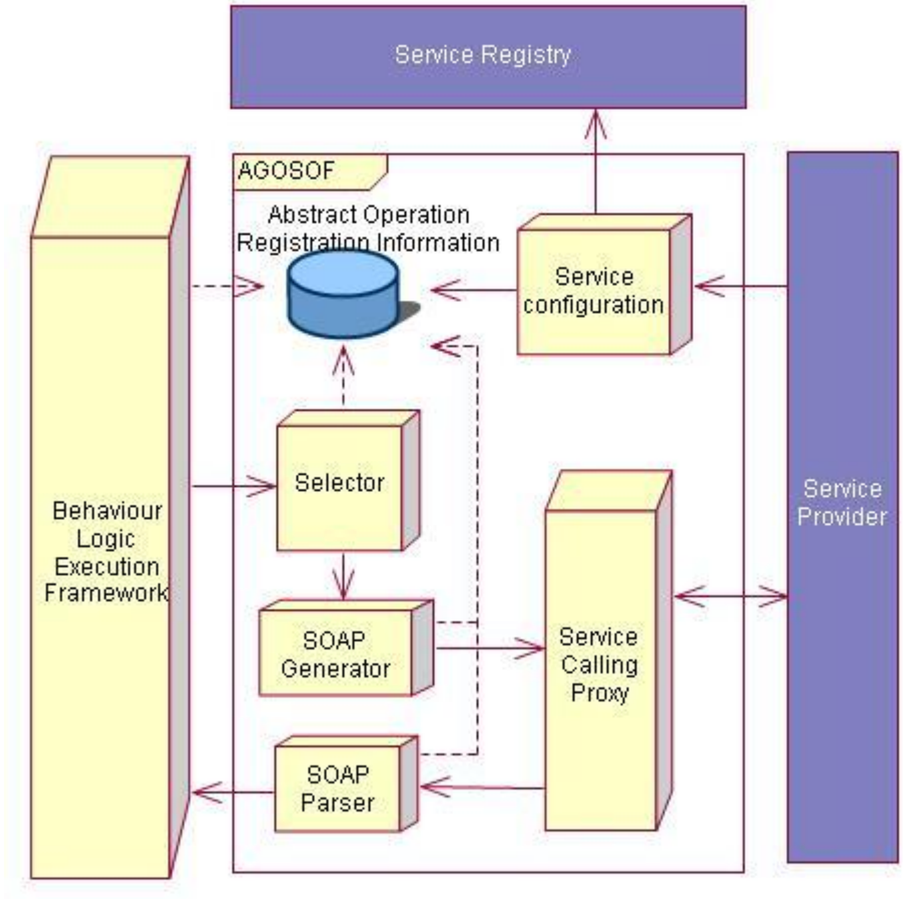


Figure 7. 14 The Structure of AGOSOF

AGOSOF is composed of Abstract Operation Registration Information, Service Configuration, Service Calling Proxy, Selector, SOAP Generator and SOAP Parser.

➢ Abstract Operation Registration Information

Abstract Operation Registration Information is generated by DSMEI in the process of parsing and compiling xDSM application model. It is both web service specification and message specification of AGOSOF. It contains all abstract operation information of xDSM models, and the basic information and runtime information of the relevant web services. Abstract Operation Registration Information is a structured dataset:

Table 7. 2 Abstract Operation Registration Information

ModelID		Identification of a xDSM model. It is the namespace of Abstract Operation
AbstractOperation	Name	name of abstract operation
	InputPin	InputPin of abstract operation
	OutputPin	OuputPin of abstract operation
ServiceCount		The number of web services which realise abstract operation
Service0	Name	Opration Name of web service
	URL	URL address of web service
	Protocol	Binding protocol of web service
	SOAP	SOAP binding information corresponded to web service
	WSDL	WSDL description of web service
	InputMap	Document matching script information of service input
	OutputMap	Document matching script information of service output
	ResponseTime	Runtime information, the average response time of service
	Loaded	Runtime information, load amount of service call
ServiceN	…..	The same as Service0

- **Service Configuration**

It is used to deploy the relevant web services cluster to abstract operations of xDSM models. It searches Service Registry or appoints Service Provider directly, gets

WSDL and selects the relevant operation from Service Provider, and defines the document matching script information of input and output for the operation of web service.

- **Service Calling Proxy**

Service Calling Proxy is the interaction proxy between AGOSOF and Service Provider. It implements SOAP specification and protocol binding according to web service standard, sends web service calling request (SOAP service calling request) and gets web service calling result (SOAP service result return, SOAP service fault return) in order to make AGOSOF call web service transparently.

- **Selector**

Selector selects and calls the most optimistic web service in the light of the values of ResponseTime and Loaded in accordance with Abstract Operation Registration Information.

- **SOAP Generator**

In accordance with Abstract Operation Registration Information, SOAP Generator parses the selected abstract operation execution request message document and transforms it into SOAP service calling request document, then sends it to Service Calling Proxy.

- **SOAP Parser**

In accordance with Abstract Operation Registration Information, SOAP Parser parses SOAP service result return document received by Service Calling Proxy and transforms it into abstract operation execution return message document, then sends it to BLEF.

7.5.2 AGOS Service Configuration

DSMEI generates Abstract Operation Registration Information in the process of compiling and parsing xDSM application model. Abstract Operation Registration Information is not insufficient at present. It is only with the basic information of abstract operation. Service Configuration appoints the relevant web service cluster for the abstract operation, provides matching script information of service input and ouput, and generates the complete web service specification and message specification so as to make AGOSOF working well. Meanwhile, AGOS system can be configured dynamically by Service Configuration at runtime so that web services are updated online.

Service Configuration extracts the WSDL description of the appointed service and selects the relevant operation to form the corresponding information between abstract operation and service in detail:

- Name: to extract the attribute of *Name* of Operation of the element of Binding from WSDL

- URL: to extract the attribute of *Location* of soap:address of the element of Service from WSDL

- Protocol: to extract the attribute of *Transport* of soap:binding of the element of Binding from WSDL

- SOAP: to extract the input and output message of Operation of the element of Binding from WSDL, which contains SOAP format request, InputPin and OutputPin

- WSDL: WSDL text message

If the InputPin sequence or the OutputPin sequence of the selected web service is different from that of the abstract operation, the Pin sequence and SOAP parameter sequence matching should be taken in Service Configuration.

The matching operation is defined by scripts. Target business document is gotten by executing the scripts while source business document inputs. The matching scripts can do some simple operations, such as calculating, evaluating, and XML object operation. The matching scripts between the InputPin sequence of abstract operation and SOAP service calling document message element is saved in InputMap. The matching scripts between SOAP service result return document message element and the OutpuPin sequence of abstract operation is saved in OutputMap.

7.5.3 AGOS Service Call

AGOS service call is a procedure that DSMEI invokes web service to implement abstract operation. It is the standard web service call by SOAP protocol for users. It is a SOAP business document exchange process in DSMEI which sends SOAP service calling request documents and receives SOAP service result return documents to implement AGOS call. Meanwhile, the exchange between Message business documents and SOAP business documents is also carried out in the process of abstract operation call. Web services are used to implement computational logic of abstract operation. The AGOS service calling procedure in AGOSOF is as follows:

1. BLEU sends abstract operation execution request message when interpreting and parsing to abstract operations. ECU transfers it as document to Selector.

2. Selector extracts the abstract operation information from abstract operation execution request message document, finds the matched AbstractOperation with the relevant ModelID from Abstract Operation Registration Information, and queries the runtime information of the relevant web services (Response Time,

Loaded). The web service with the minimum Min value is selected in the light of Min (ResponseTime * (Loaded + 1)). Meanwhile, abstract operation execution request message document and the target web service information are transferred to SOAP Generator.

3. SOAP Generator parses abstract operation execution request message document, and get InputMap from Abstract Operation Registration Information according to the target web service information. It executes the matching scripts and transforms the InputPin sequence of abstract operation into the message elements of SOAP service calling document. Then it integrates SOAP binding information, generates SOAP service calling request document, and sends it to Service Calling Proxy with the target web service URL and Protocol information.

4. Service Calling Proxy binding SOAP service calling request document in the light of the target web service URL and Protocol information, and sends web service calling request message to Service Provider. Service Calling Proxy sends the result return document and the target web service to SOAP Parser when it gets SOAP service result return document. Response Time and Loaded information matched with abstract operation are updated into Abstract Operation Registration Information before or after web service call.

5. SOAP Parser parses SOAP service result return document, and gets OutputMap from Abstract Operation Registration Information according to the target web service. It executes the matching scripts and transforms SOAP service result return document into the OutputPin sequence of abstract operation. Abstract operation execution return message document is generated and sent to ECU then to BLEU to implement abstract operation invoking.

SOAP business document exchange is carried out by Service Calling Proxy in AGOS service call. Meanwhile, Selector, SOAP Generator and SOAP Parser compose

Exchange Pipeline which implements the mapping and matching between Message business document and SOAP business document with Abstract Operation Register Information as web service specification and message specification. Therefore, abstract operations are implemented by DSMEI with web services.

7.5.4 AGOS Service Virtualisation

Service virtualisation is adopted as an available communication method for service users and providers. It provides a simple way for users to utilise the dynamic and distributed network service resources, and implements service deployment and update dynamically [31]. The technical details can be encapsulated (such as web service binding protocol, accessing mode).

Service virtualisation is a kind of important technique for building the service oriented flexible framework. The aim of service virtualisation is to reduce the complexity of service utilisation, and provide the simpler calling mode. AGOSOF achieves service virtualisation by web service model based on business document exchange. It can support service location transparency and service transparent migration. It raises the dynamic ability of web service (dynamic matching, dynamic binding, dynamic updating, dynamic deploying). AGOS service virtualisation is shown on two aspects as Figure 7.15.

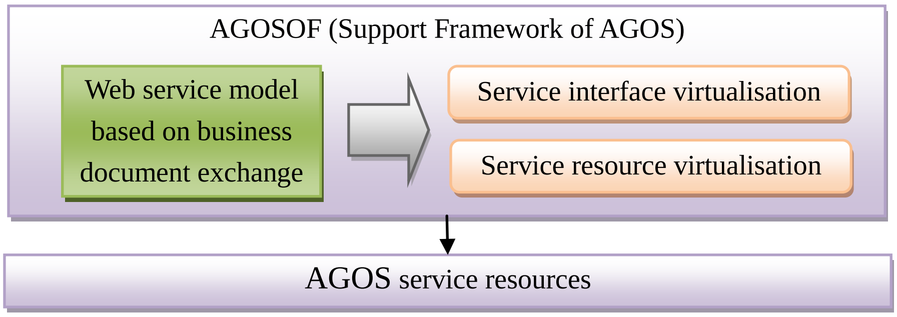

AGOSOF (Support Framework of AGOS)
Web service model based on business document exchange
Service interface virtualisation
Service resource virtualisation
AGOS service resources

Figure 7. 15 Service Virtualisation of AGOS

- Service interface virtualisation: as the implementation entities of abstract operations, web service interfaces may be different from abstract operation interfaces. Dynamic adaptation can be achieved by executing the matching scripts to transform between the Pin sequence of abstract operations and SOAP documents so as to implement web service interface virtualisation.

- Service resource virtualisation: an abstract operation can correspond with multiple web services as its implementation entities. The web service dynamically binding technique is adopted to do load balancing for multiple service entities and select the most optimistic service entity in the light of the result. Furthermore, abstract operation service configuration can be changed, and web services can be updated and deployed online to implement web service resource virtualisation.

Service virtualisation of AGOS plays an important role in DSMEI. Firstly, it can shield the change and update of web services for the behaviour logic infrastructure, and provide simple and identical interfaces, and make DSMEI be able to utilise the complex, dynamic and easily changed web service resources at the low level by a simple and stable way. Secondly, it enhances the flexibility of service implementation and deployment for service providers. Service providers can implement and deploy web services following the established requirements which will not affect AGOS utilisation by DSMEI. So web services can be used openly, and web services resources is available for reuse.

7.6 Summary

xDSM models cannot be executed directly. It depends on the execution environment to be interpreted and executed.

In this chapter, DSMEI is designed and instantiated as the execution environment for xDSM models which is parsed into operation sequences with accurate semantic, and the operations is executed to realise the application system. DSMEI integrates domain framework and combines AGOS to provide software functional entities for the virtual operations, which makes xDSM application model become the executable software product. Model is executed indeed so as to realise MDD.

DSMEI is composed of three frameworks: one internal framework -- BLEF; and two external frameworks -- DSPROF and AGOSOF. They work together to parse and execute xDSM models to achieve system objectives.

- BLEF parses behaviour semantics of xDSM application model, transforms them into the operational sequence with accurate semantics.

- DSPROF provides the xDSM model execution application interfaces to end users by the way of web services.

- AGOSOF adopts web services as software entities for xDSM model execution.

- Web services model based on business document exchange is proposed to design and realise DSPROF and AGOSOF for xDSM model execution. On one hand, the dynamic publishing and calling of domain application web services are realised; on the other hand, the virtualisation of AGOS services is realised.

- AGOS is not only provided as software functional entities for xDSM model execution, but also served as software assets for large-scale reuse.

Chapter 8

Domain-Specific Modelling Process and Implementation Framework

8.1 Domain Space

Domain space [130] is the set of domain entities and the relationships defined on the set of domain entities. Based upon domain analysis, domain space extracts the typical features of domain concepts to construct the set of domain entities and the relationships on the entity set. Domain entities are constructed according to the domain-specific architecture standard.

SODSMI approach is organised by domain space at architectural level which is the elementary unit of the domain-specific modelling and implementation framework. Domain space involves the problem space and the solution space of a specific domain.

Domain space integrates domain framework on the basis of xDSM meta-model, which is the integrated representation of domain-specific knowledge and its implementation. Domain-specific software reuse at architectural level can be achieved by reusing and assembling domain spaces. Domain space expands the scale and scope of domain-specific models and their implementation.

8.1.1 Architecture of Domain Space

In SODSMI, domain space is the elementary unit of the domain-specific modelling and implementation. It involves domain-specific architecture standards. Domain space integrates service information of AGOS on the basis of xDSM meta-model, which represents domain-specific knowledge and its realisation. In addition, domain space involves domain public behaviours and domain space dependences to support the reference and composition of domain spaces, which realises software reuse at architectural level.

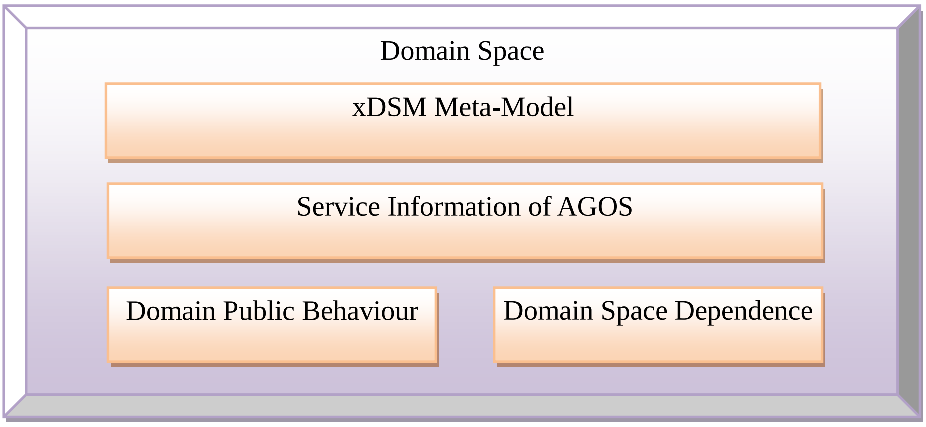


Figure 8. 1 Architecture of Domain Space

- xDSM meta-model: xDSM meta-model for a specific domain is constructed with XDML language in GME. The name and version of the domain space come from xDSM meta-model. xDSM meta-model is the core of the domain space, which represents a specific domain concept. Domain concepts can be concrete or abstract. On one hand, model elements, constraints and behaviours of xDSM meta-model come from the problem space directly; on the other hand, the domain framework corresponding with xDSM meta-model represents the solution space. By the support of AGOS, xDSM application model constructed upon xDSM meta-model can be executed directly in DSMEI, and generate the domain-specific application instance. Therefore, xDSM meta-model represents domain knowledge from the problem space and the solution space.

- Service Information of AGOS: AGOS collects web services as software functional entities for executing xDSM models. AGOS is associated with abstract operations of xDSM model to generate AGOS service configuration in DSMEI so as to realise the virtualised operations. Domain space involves service information of AGOS as the system implementation entities of the solution space, including web services information configuration and interface matching configuration.

- Domain Public Behaviour: domain space supplies domain public behaviours at architectural level for domain software reuse and combination. Service interfaces offered by domain space can be used by other behaviour scenarios. Domain space constructed domain-specific public behaviours though its xDSM meta-model. All domain public behaviours contained by the domain space are optional, and can be expanded according to the requirements of combination.

- Domain Space Dependency: there are dependencies between the domain spaces when they are assembled. Child domain space depends on parent domain space. In other words, child domain space and its domain framework are constructed on the basis of parent domain space.

Domain space is a self-sufficient architecture which contains domain-specific concepts and the corresponding entities to support xDSM application modelling and model executing independently. xDSM application model is constructed based upon xDSM meta-model of the domain space, and transformed into the service-oriented domain-specific application by the support of AGOS service information. In addition, domain space provides other domains with service interfaces of the specific domain via domain public behaviours, which encapsulate the concrete implementation and data.

Domain space is also an open architecture due to the dynamic reconstructable and extensibility of models. xDSM meta-model and the constructed application model can be reconstructed in GME. The reconstruction process is visual and controllable. Furthermore, the complex xDSM application model can be constructed with the set of xDSM meta-models formed by domain space dependency.

8.1.2 Reuse and Composition of Domain Spaces

Domain engineering roots in software reuse [107, 2]. Domain-specific modelling is to construct and realise the different application models in a specific domain for software reuse. Domain-specific modelling is based upon the reusable infrastructure which makes up of domain-specific meta-model and domain framework.

In general, each domain-specific meta-model and its domain framework follow to the development pattern of “Everything starts from nothing.” Domain-specific modelling aims at solving software development problems existing in a certain specific application domain. Properly speaking, domain-specific modelling is fit for the small-scale specific domain to construct its meta-model. On the contrary, the big-scale specific domain will result in huge domain-specific meta-model and domain framework. According to “7+2 principle [78]”, it could be better for developers if modelling elements can be controlled within 9 at a time, otherwise it will make developers hard to understand the model. So it is necessary to plot out the hierarchy and the granularity for the domain space of a certain domain. For instance, the domain of office automation can be divided into the domain of document management and the domain of authority management.

At the same time, the domain space can be reused adequately. The new domain spaces can be constructed with the composition hierarchy of domain spaces incrementally increasing to achieve the reuse at domain level.

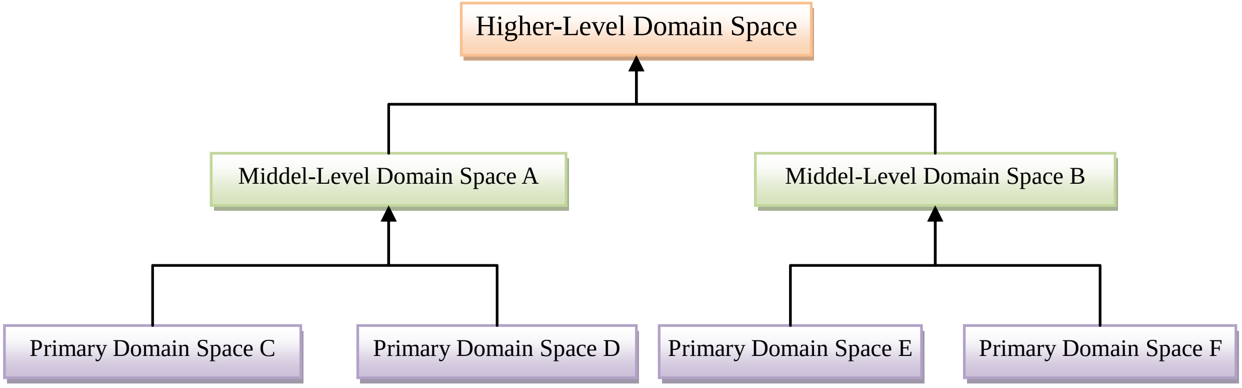


Figure 8. 2 Hierarchy of Domain Spaces

The higher-level domain space is compounded by the primary domain space with different granularities at different levels. The composition of domain spaces not only raises the abstract level but also extends the functionalities scale of model concepts in domain space. It is no limit to the composition levels so that the existing domain spaces can be incrementally compounded to reuse domain knowledge.

It is necessary to divide the domain into small ones while constructing the domain space of a large specific domain. The large specific domain is considered as one being composed of a series of domain spaces with high cohesive. Each domain space contains its domain-specific meta-model and domain framework. They must satisfy the following rules:

- Single domain: domain concepts and contents of a domain space that is just for a specific domain, not shared with multiple specific domains.

- Common reuse：Domain space is responsible for all domain spaces that depend on it base on high-cohesion. Changes of the referred domain spaces do not affect child domain spaces.

- Non-cyclic dependency: it means that there is no cycle dependency among domain spaces.

Domain spaces defined by SODSMI with well-structured architecture can be reused and compounded to form the domain space hierarchy in GME and DSMEI.

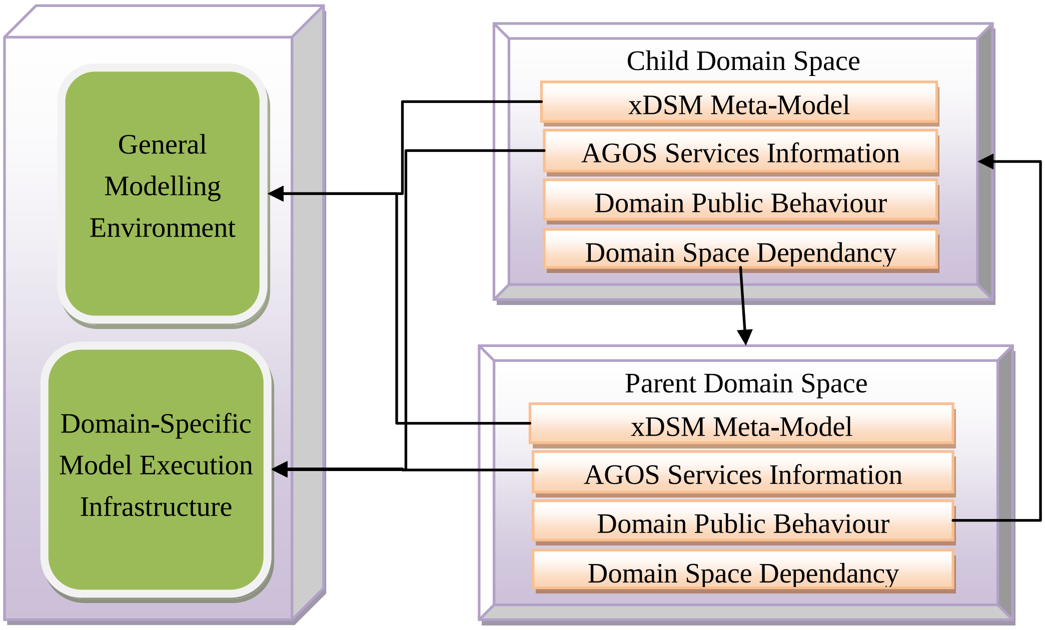


Figure 8. 3 Reuse and Composition of Domain Spaces

- ➢ xDSM meta-model describes concepts and rules of the specific domain. The entire xDSM meta-model is loaded into GME to generate the visual xDSM application modelling environment. The definition of xDSM meta-model provides architectures, concepts and constraints for the domain space composition. The target meta-model is constructed with the xDSM meta-model of the parent domain space in GME by the extension mechanism of xDSM meta-model.

- ➢ AGOS service information of the domain space provides DSMEI with AGOS service configuration. It loads abstract operations as registration information, and takes xDSM meta-model of the domain space as the benchmark to provide domain framework and the supporting services for model execution. AGOS service information of different domain spaces is loaded one by one into DSMEI irrepeatedly to support the execution of the compounded xDSM application model.

- Child domain space can call the domain public behaviours of parent domain space, and utilise the domain business functionalities of the parent domain space without application modelling.

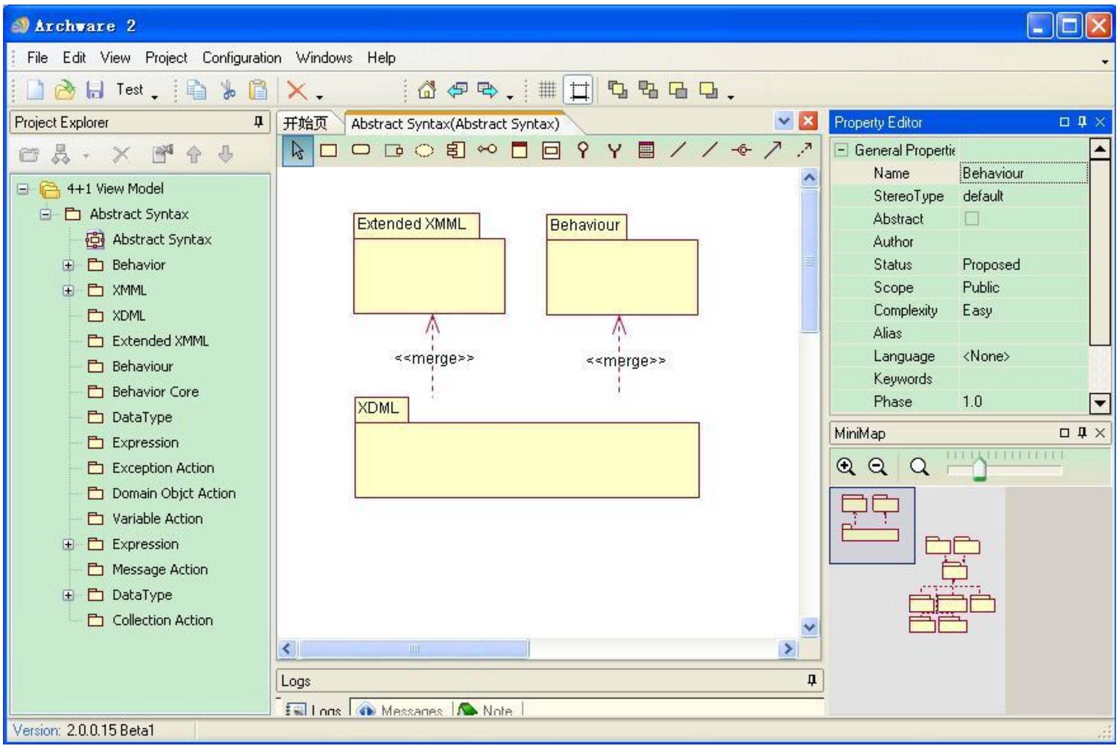
- The domain space dependency of child domain space keeps the information of parent domain space in order to provide the dependent relationship for domain space composition.

The reuse and composition of domain spaces provide the modelling mechanism and the extension mechanism for xDSM. Domain space reuse is systematic reuse from modelling elements to domain framework. It raises software reuse to a new level -- domain knowledge reuse at domain level. By reusing and compounding domain spaces, the domain space with larger scale can be constructed incrementally which improves the efficiency and quality of domain-specific modelling significantly.

8.2 Domain-Specific Modelling Process

The domain-specific modelling process is accomplished by building xDSM meta-model and xDSM application model in GME.

GME we developed is Archware, supporting XDML language and carrying out domain-specific modelling. Archware was extended and added new functions for behaviour modelling and xDSM model definition. The core functions focus on two aspects: the first is to provide an xDSM meta-modelling environment in which domain space can be created, loaded and outputted. The second is that it can parse xDSM meta-model, generate the supporting environment for xDSM application modelling, and support xDSM application modelling.



Name	Behaviour
StereoType	default
Abstract	□
Author	
Status	Proposed
Scope	Public
Complexity	Easy
Alias	
Language	<None>
Keywords	
Phase	1.0

Figure 8. 4 Generic Modelling Environment -- Archware

The domain-specific modelling process is an iterative process. The process includes the following steps, from domain analysis to xDSM application modelling.

- Domain Analysis

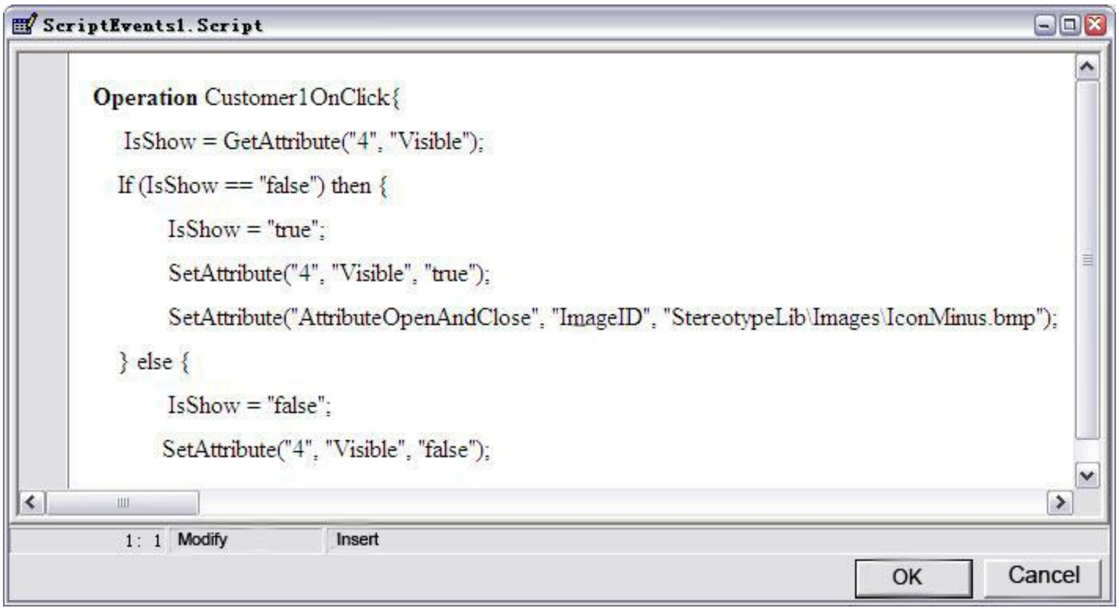
Domain analysis aims at identifying domain boundary and extracting domain concepts and the relationships between concepts. Domain analysis collects the common requirements of domain systems, finds the similarity and differences from application systems, and describes the architectural model which is suitable for all application systems in the specific domain. Domain experts study the developing domain-specific system, identify and capture the similar information from domain systems. By mining internal features and rules, domain experts sort out and organise the information to get the corresponding domain concepts and their relationships so as to identify the boundary of domain finally.

- To create domain space and xDSM meta-model

Domain space is the set of domain entities and the relationships defined on the set of domain entities. The core of domain space is xDSM meta-model. When domain space is created by developers, the corresponding xDSM meta-model project is created too. If there has been the relevant domain space of the specific domain, the domain space just needed to load in Archware. Archware adopts the visual method to define attributes, behaviours, events, constraints, rules and diagrams of model elements so as to get xDSM meta-model.

- Attribute definition: to define all attributes of model elements. Archware provides Attribute Form Designer for developers, which can be used to design attribute editor for the model elements with complex attributes.

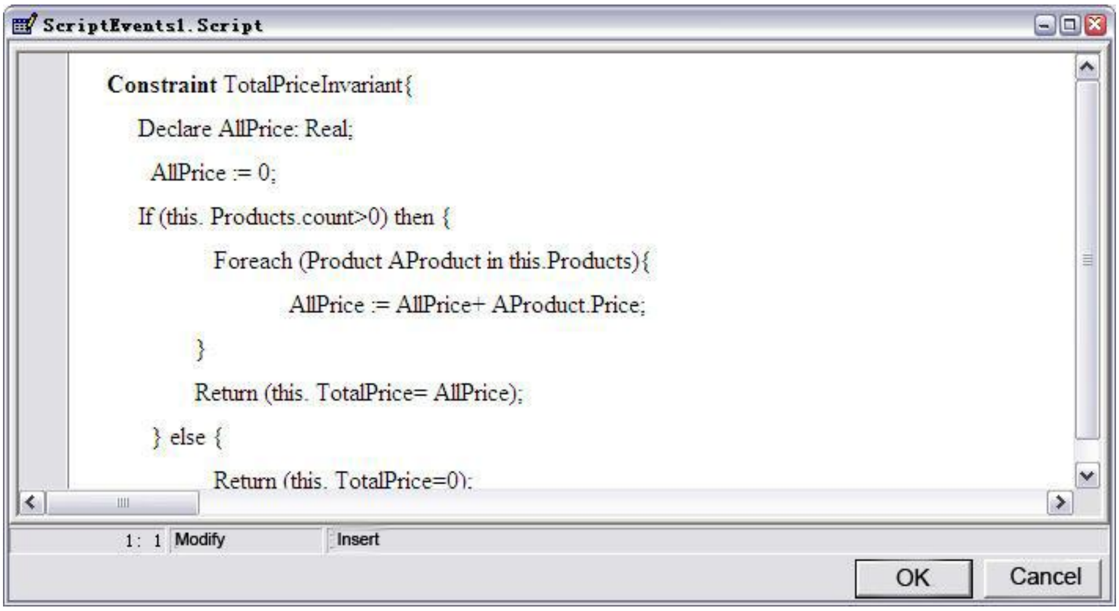
- Behaviour and event definition: to define behaviours and events of model elements. Behaviour of model elements is described by behaviour scenario or action specification. The way to describing behaviour scenario is as same as xDSM application modelling. In Archware, behaviour scenario is defined via the primary xDSM meta-model to describe the behaviours of model elements. AS&MC Editor is used to edit action specifications according with AS&MC syntax to describe the behaviours of model elements. And event definition of model elements is also described by action specifications according with AS&MC syntax which is edited in AS&MC Editor, as shown in the following figure.



```
Operation Customer1OnClick{
   IsShow = GetAttribute("4", "Visible");
  If (IsShow == "false") then {
        IsShow = "true";
        SetAttribute("4", "Visible", "true");
        SetAttribute("AttributeOpenAndClose", "ImageID", "StereotypeLib\Images\IconMinus.bmp");
  } else {
        IsShow = "false";
       SetAttribute("4", "Visible", "false");
```

Figure 8. 5 Action Specification Defined in AS&MC Editor

- Constraint definition: to define constraints of model elements or attributes and behaviours of model elements. In Archware, AS&MC Editor is used to edit model constraints according with AS&MC syntax to describe the relevant constraints of model elements.

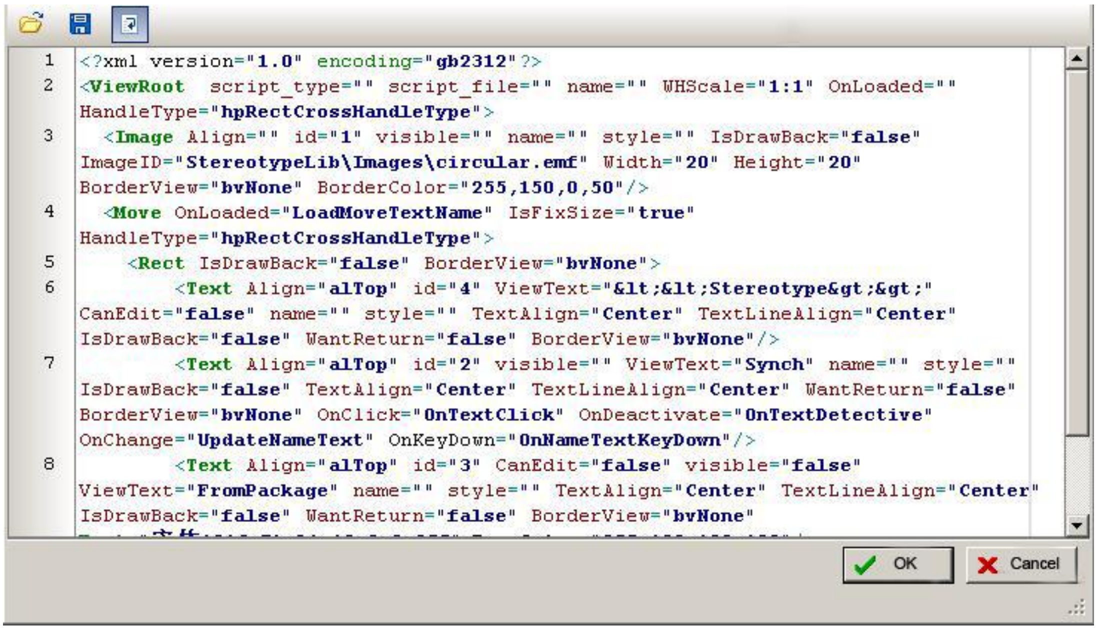


```
Constraint TotalPriceInvariant{
   Declare AllPrice: Real;
    AllPrice := 0;
   If (this. Products.count>0) then {
          Foreach (Product AProduct in this.Products){
                 AllPrice := AllPrice+ AProduct.Price;
        }
        Return (this. TotalPrice= AllPrice);
    } else {
          Return (this. TotalPrice=0);
```

Figure 8. 6 Model Constraint Defined in AS&MC Editor

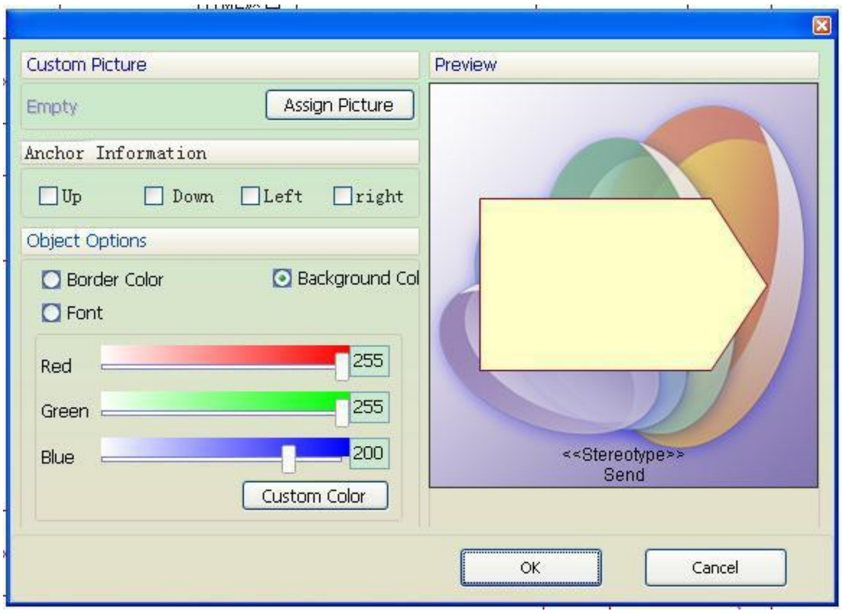
- Rule definition: to define modelling rules of model elements. The rules include association rule, refinement rule and reference rule of model elements. The definition is made by meta-configuration manager of Archware.

- Diagram definition: to define the visual meta-graphic of model elements. Element meta-graphic designer in Archware is used to design the corresponding visual meta-graphic for each element of the meta-model. There are two important parts of element meta-graphic designer: one is meta-graphic appearance description code editor, the other is meta-graphic appearance preview and configuration form. The former is a tool similar to HTML editor, and its design result could be previewed in the latter that also provides the function to adjust the appearance style of model elements.



```
<?xml version="1.0" encoding="gb2312"?>
<ViewRoot  script_type="" script_file="" name="" WHScale="1:1" OnLoaded=""
HandleType="hpRectCrossHandleType">
  <Image Align="" id="1" visible="" name="" style="" IsDrawBack="false"
ImageID="StereotypeLib\Images\circular.emf" Width="20" Height="20"
BorderView="bvNone" BorderColor="255,150,0,50"/>
  <Move OnLoaded="LoadMoveTextName" IsFixSize="true"
HandleType="hpRectCrossHandleType">
    <Rect IsDrawBack="false" BorderView="bvNone">
        <Text Align="alTop" id="4" ViewText="&lt;&lt;Stereotype&gt;&gt;"
CanEdit="false" name="" style="" TextAlign="Center" TextLineAlign="Center"
IsDrawBack="false" WantReturn="false" BorderView="bvNone"/>
        <Text Align="alTop" id="2" visible="" ViewText="Synch" name="" style=""
IsDrawBack="false" TextAlign="Center" TextLineAlign="Center" WantReturn="false"
BorderView="bvNone" OnClick="OnTextClick" OnDeactivate="OnTextDetective"
OnChange="UpdateNameText" OnKeyDown="OnNameTextKeyDown"/>
        <Text Align="alTop" id="3" CanEdit="false" visible="false"
ViewText="FromPackage" name="" style="" TextAlign="Center" TextLineAlign="Center"
IsDrawBack="false" WantReturn="false" BorderView="bvNone"
```

Figure 8. 7 Meta-Graphic Appearance Description Code Editor



Custom Picture

Empty

Anchor Information

□ Up □ Down □ Left □ right

Object Options

○ Border Color ● Background Col

○ Font

Red 255

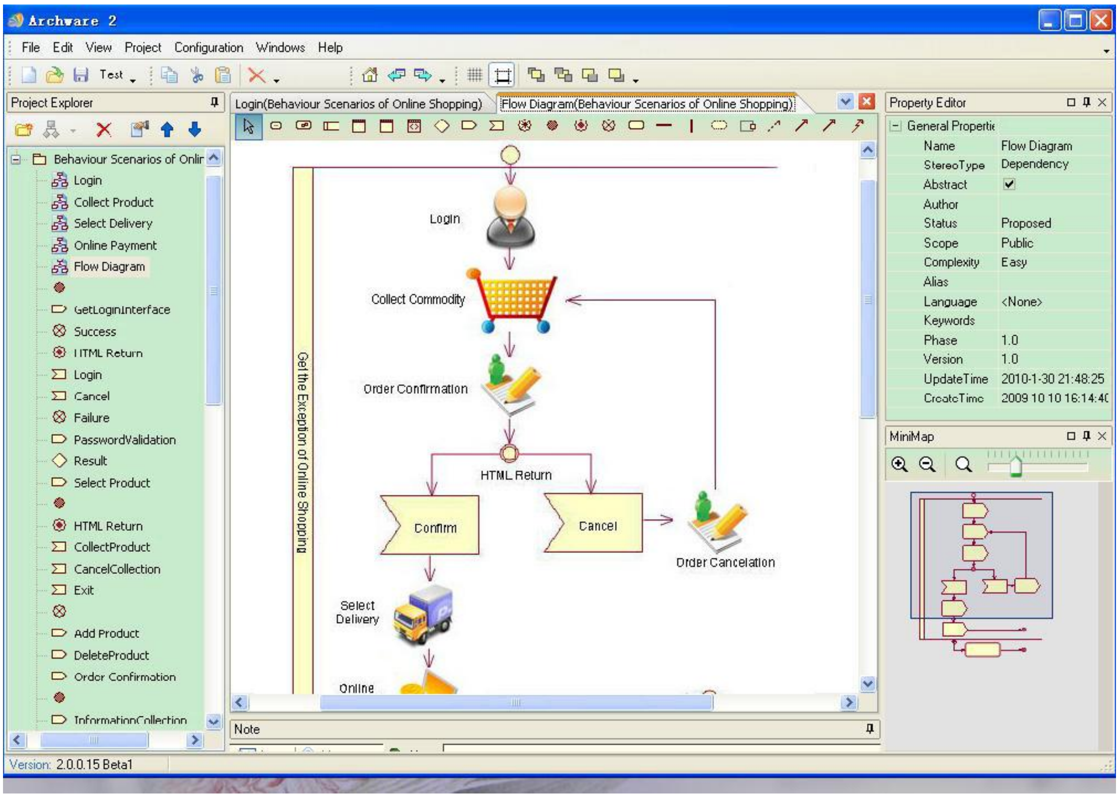
Green 255

Blue 200

Figure 8. 8 Meta-Graphic Appearance Preview and Configuration Form

- ➢ To create xDSM application model

End users create xDSM application model on the basis of xDSM meta-model in Archware as follows: The first is to analyse and extract all system objectives according to the requirements specifications of the target application system. The second is to create behaviour scenario for each system objective. The third is to describe system behaviours by using model elements of the xDSM meta-model to achieve system objectives. During the process of application modelling, Archware will execute the modelling rules and constraints defined by the xDSM meta-model, as well as control and instruct modelling activities. xDSM application modelling is shown in Figure 8.9.

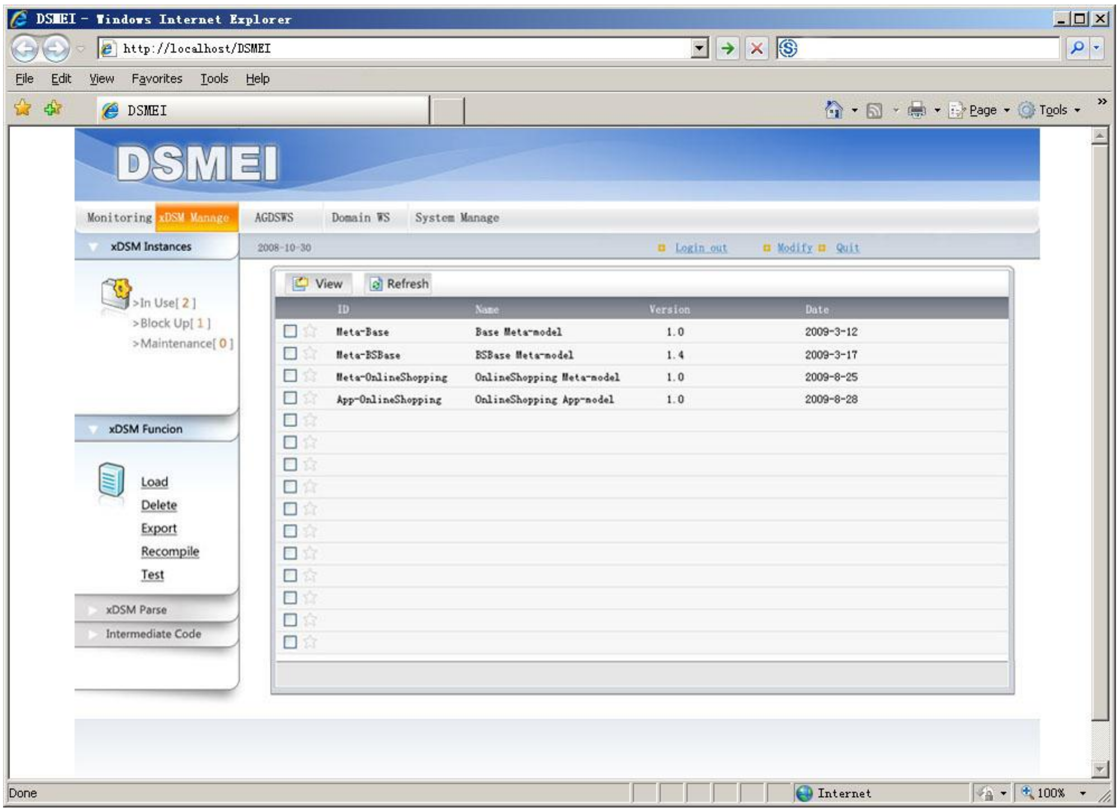


Name	Flow Diagram
StereoType	Dependency
Abstract	☑
Author	
Status	Proposed
Scope	Public
Complexity	Easy
Alias	
Language	<None>
Keywords	
Phase	1.0
Version	1.0
UpdateTime	2010-1-30 21:48:25
CreateTime	2009 10 10 16:14:4(

Figure 8. 9 Application Modelling in Archware

8.3 Domain-Specific Implementation Framework

The domain-specific implementation framework is an instruction for domain-specific model implementation. The core is DSMEI which takes web service as software function input and output entities, interprets and executes xDSM application model to complete domain-specific model implementation. There are three core functions of DSMEI: the first is to compile xDSM application model, parse and execute the intermediate code to accomplish the behaviour logic described by xDSM application model. The second is to integrate domain framework with AGOS in order to provide execution environment for xDSM application model. The third is to output the execution result of xDSM application model by domain application web services calling mechanism to achieve system implementation.



	ID	Name	Version	Date
□ ☆	Meta-Base	Base Meta-model	1.0	2009-3-12
□ ☆	Meta-BSBase	BSBase Meta-model	1.4	2009-3-17
□ ☆	Meta-OnlineShopping	OnlineShopping Meta-model	1.0	2009-8-25
□ ☆	App-OnlineShopping	OnlineShopping App-model	1.0	2009-8-28
□ ☆				
□ ☆				
□ ☆				
□ ☆				
□ ☆				
□ ☆				
□ ☆				
□ ☆				
□ ☆				
□ ☆				
□ ☆				

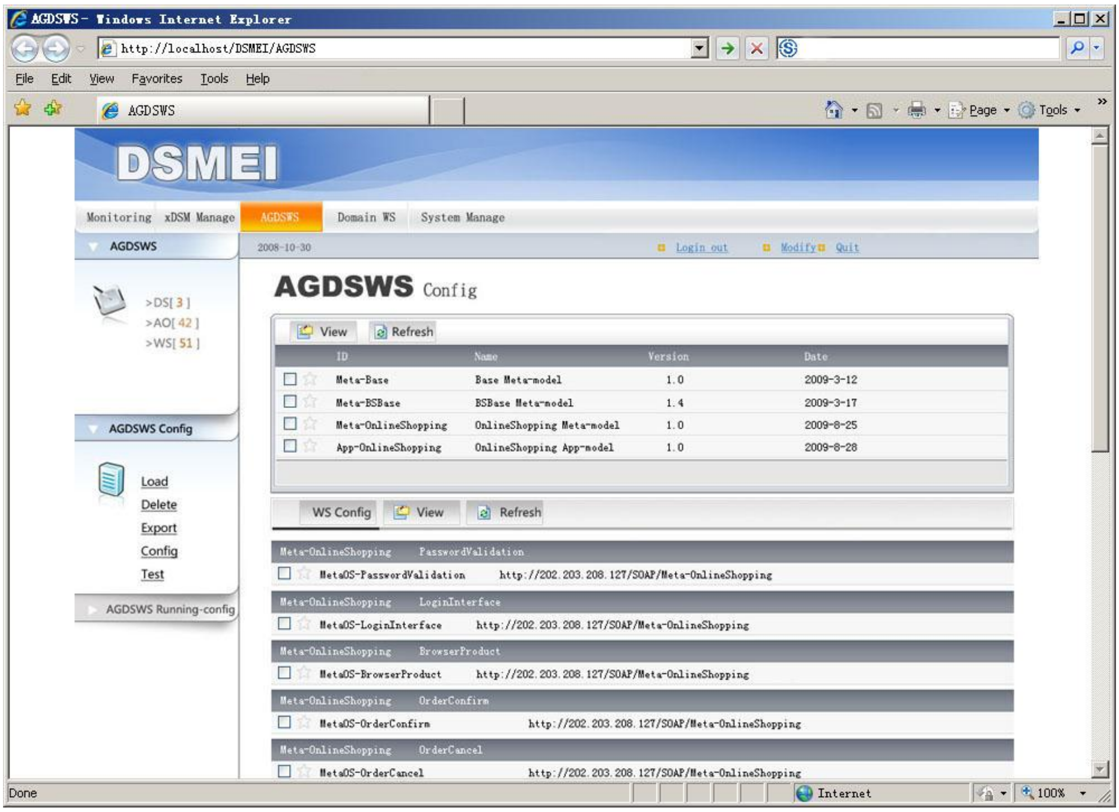
Figure 8. 10 DSMEI Console

The domain-specific modelling implementation framework mainly involves the following steps:

- To develop and configure domain-specific supporting services

After creating xDSM meta-model, domain experts can provide the detail of abstract operations including operation name, Pin and the detailed function declaration. The design and implementation of abstract operations are carried out by technical experts with web services. Technical experts may also reuse the existing web services to realise abstract operations. If it is required, technical experts will write the matching scripts of input and output documents.

This process is a part of xDSM meta-modelling, and accomplished by domain experts and technical experts from the same organisation to ensure the consistency of xDSM meta-model and domain-specific supporting services. It forms the corresponding information between abstract operations and domain-specific supporting services which is AGOS service information, and adds it to the domain space. When DSMEI compiles the xDSM application model, AGOS service information will be loaded to complete AGOS service configuration automatically, and complement Abstract Operation Registration Information. When there are requirements of changing the relevant implementation of abstract operation, it just need adjust the domain-specific supporting service information though AGOS service information configuration of DSMEI -- web services should be dynamically updated online.



	ID	Name	Version	Date
□	Meta-Base	Base Meta-model	1.0	2009-3-12
□	Meta-BSBase	BSBase Meta-model	1.4	2009-3-17
□	Meta-OnlineShopping	OnlineShopping Meta-model	1.0	2009-8-25
□	App-OnlineShopping	OnlineShopping App-model	1.0	2009-8-28

Meta-OnlineShopping PasswordValidation
□ MetaOS-PasswordValidation http://202.203.208.127/SOAP/Meta-OnlineShopping

Meta-OnlineShopping LoginInterface
□ MetaOS-LoginInterface http://202.203.208.127/SOAP/Meta-OnlineShopping

Meta-OnlineShopping BrowserProduct
□ MetaOS-BrowserProduct http://202.203.208.127/SOAP/Meta-OnlineShopping

Meta-OnlineShopping OrderConfirm
□ MetaOS-OrderConfirm http://202.203.208.127/SOAP/Meta-OnlineShopping

Meta-OnlineShopping OrderCancel
□ MetaOS-OrderCancel http://202.203.208.127/SOAP/Meta-OnlineShopping

Figure 8. 11 Service Information Configuration of AGOS

- To compile, interpret and execute xDSM application model

xDSM application model described by XDML language cannot be executed directly. DSMEI extracts behaviour scenarios of xDSM application model and compiled them into the intermediate code, then interprets and executes the intermediate code to achieve behaviour logic defined by xDSM application model. The details can be found in Chapter 7.

- To call and provide web services

DSMEI provides open application interfaces of xDSM model execution for end users by web services, and employs web services as the core functional implementation entities of xDSM execution. DSMEI provides the mechanisms of calling and providing web services separately to support xDSM model execution. The details can be found in Chapter 7.

8.4 Summary

In this chapter, the domain-specific modelling process and implementation framework are introduced. Domain space is proposed as the elementary unit of the domain-specific modelling process and implementation framework. The reuse and composition of domain spaces are realised by the flexible architecture of domain space on the framework of SODSMI. It makes software reuse at domain level, realises the reuse of domain knowledge, and openly extends the range and scale of domain-specific models.

The domain-specific modelling process includes from domain analysis, to xDSM meta-modelling, and xDSM application modelling. It is an iterative process.

The domain-specific implementation framework is an instruction for supporting the model implementation. The core is DSMEI which takes web service as software functional input and output entities, interprets and executes xDSM application model to complete the model implementation.

Chapter 9

Case Studies

9.1 Overview

The SODSMI approach focuses on xDSM modelling which is accomplished by building xDSM meta-model and xDSM application model in GME. In this chapter, two case studies will be used to illustrate xDSM modelling and implementation.

The xDSM modelling process includes the following steps, from domain analysis to xDSM meta-modelling, then to xDSM application modelling. The xDSM model is constructed in Archware, and the xDSM application model is executed in DSMEI to realise the software system.

9.2 Conference Registration System based on Mobile

In the section, conference registration system based on mobile is designed as a case study of xDSM modelling. Domain analysis, meta-modelling and application modelling are carried out to show the feasible domain-specific modelling.

Conferences usually adopt online registration for participants to register their relevant information. However, it will be more convenient if online registration can be completed with mobiles. Conference registration system based on mobile is designed mainly for participants to register online with their mobiles. The system functions are described as follows:

➢ Function tips: After logining the system successfully, users can see the welcome page and function tips such as registration, conference schedule browsing and registration cancellation.

- Registration: Users are prompted to enter his/her name and password, and select the way to pay the costs associated with the conference. Users exit registration if the operation is successful. The system sends text message to users to show the registration is successful.
- Conference schedule browsing: Only the registered users can browse the relevant conference information, such as scheduling.

9.2.1 Domain analysis

Conference registration system based on mobile mainly involves the following functions, conferee registration, schedule browsing, relevant information prompting and text message sending. It is similar to other application systems based on mobile, for instances, restaurant reservation system based on mobile and hotel reservation system based on mobile.

The registration system based on mobile is taken as a specific space to carry out domain analysis. The following main domain concepts are extracted:

- Conferee: as the main part of the registration system based on mobile, Conferee contains the related information of registered users. Conferee logins system via password checking. The other procedures are associated with Conferee, for examples, schedule browsing and payment.
- Note: it is used to show the prompt information, for examples, welcome to the system, registration is made, etc.
- Popup Menu: the function menu will be popped for the user to choose when the user presses hot key on the mobile, for example, conference registration,

conference schedule browsing, etc.

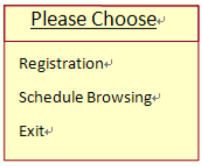
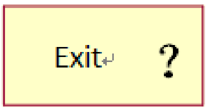
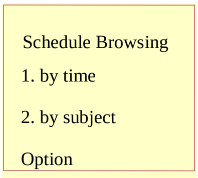
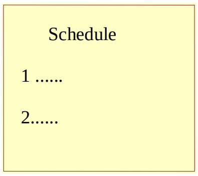
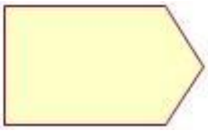
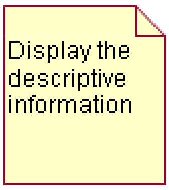
- Query: it requests users to input their information, for examples, user name, user exit, etc.
- List: it shows the optional items, for examples, select registration, conference schedule browsing, and cancel registration.
- Form: it shows information and the related operations.
- Text Message: it is sent to the relevant conferee.
- Comment: it shows the descriptive information.

9.2.2 Meta-modelling

According to the above domain analysis, the related domain concepts are extracted to define the xDSM meta-model of the registration system based on mobile. Firstly, domain entities and their attributes, icons and events of the meta-model are defined as Table 9.1.

Table 9. 1 Meta-Model Entities of the Registration System based on Mobile

Icon	Name	Attributes	Event
👤	Conferee	UserName: string PassWrd: string Registered: Boolean	[DesignTimeEvent] OnCreate
i↵	Note	Text:string	[DesignTimeEvent] ShowInfo

	Popup Menu	Prompt:string AvailableItem: string []	[DesignTimeEvent] OnSelect OnCreate
	Query	Prompt: string QueryType: string returnValue: string	[DesignTimeEvent] OnQuit OnRegistered OnPay
	List	Text: string AvailableItem: string [] Validated: Boolean	[DesignTimeEvent] OpenNewForm
	Form	Text: string Fields: string[]	[DesignTimeEvent] ShowSeletedInfo
	Text Message	Text: string Recipient: string Payment: string Returned: Boolean	[DesignTimeEvent] SendSMS
	Comment	Text:string	[DesignTimeEvent] ShowIntroInfo

The primary xDSM meta-model of the domain space of the registration system based on mobile is constructed by defining the domain entities and their attributes, icons and events in Archware. The domain entities are defined and shown in Table 9.1. Events are almost the design time events which are implemented in application modelling in Archware.

The xDSM meta-model entities of the domain space of the registration system based on mobile have many operations which carry out system functions contained in the above domain concepts. They are defined as follows:

- **Conferee**

  - Password Checking [Abstract Operation]

    Operation PassWrdValidation; Abstract;

    InputPin UserName, PassWrd: string

    OutPutPin Registered: Boolean

  - Login Interface [Abstract Operation]

    Operation LoginInterface; Abstract;

    InputPin Null；

    OutputPin Result string

  - Login[Behaviour Scenario] [Active Operation]

    Operation Login; BS; Active;

    InputPin Null；

    OutputPin Result： string

  - Exit[Behaviour Scenario] [Active Operation]

    Operation Exit; BS; Active;

    InputPin Null；

    OutputPin Result： string

- **Note**

  - ShowInfo [Abstract Operation]

    Operation ShowInfo; Abstract;

    InputPin Text: string;

    OutputPin Result: Boolean

- **Popup Menu**

  - ShowAvaliableMenus [Abstract Operation]

    Operation ShowAvaliableMenus; Abstract;

    InputPin Text: string;

    OutputPin Result: Boolean

  - OnSelect [Abstract operation]

    Operation OnSelect; Abstract;

    InputPin Text: string;

    OutputPin Result: Boolean

- **Query**

  - ShowPrompt [Abstract Operation]

    Operation ShowPrompt; Abstract;

    InputPin Text: string;

    OutputPin Result: Boolean

  - GetInfo [Abstract operation]

    Operation GetInfo; Abstract;

    InputPin null;

    OutputPin Info: string

- **Browser Agenda**

  - BrowserAgenda [Behaviour Scenario]

    Operation BrowserAgenda; BS;

    InputPin BroserInfo:string;

    OutputPin Result: Boolean

- **List**

  - ShowAvailableChoices [Abstract Operation]

    Operation ShowAvailableChoices; Abstract;

InputPin Text: string;

OutputPin Result: Boolean

- OpenForm [Abstract Operation]

  Operation OpenForm; Abstract;

  InputPin Text: string;

  OutputPin Result: Boolean

➢ **Text Message**

- SendSMS [Abstract Operation]

  Operation SendSMS ; Abstract;

  InputPin Message,Recipent:String

  OutputPin Result: Boolean

➢ **Comment**

- ShowIntroInfo [Abstract Operation]

  Operation ShowIntroInfo ; Abstract;

  InputPin null;

  OutputPin introInfo

There are xDSM meta-model entities involving the operations described with behaviour scenarios in the domain space of the registration system based on mobile. For example:

➢ **Conferee**, Behaviour Scenario of *Exit* Operation

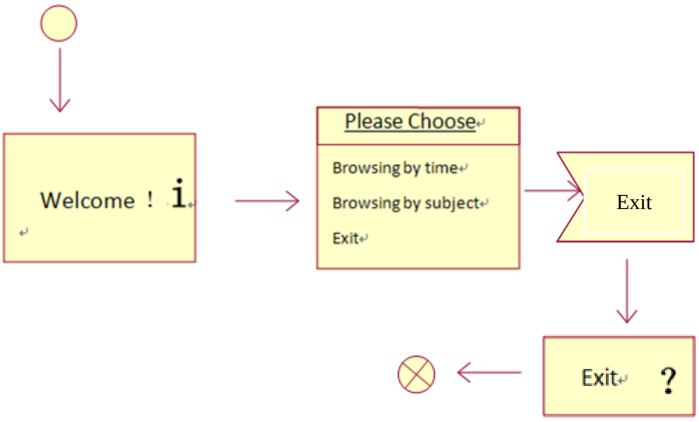


Figure 9. 1 Behaviour Scenario of *Exit* Operation

9.2.3 Application Modelling

Individual behaviour scenarios can be constructed in the phase of xDSM application modelling too, as shown in the following examples.

- **Conferee**, Behaviour Scenario of *Conference Registration*

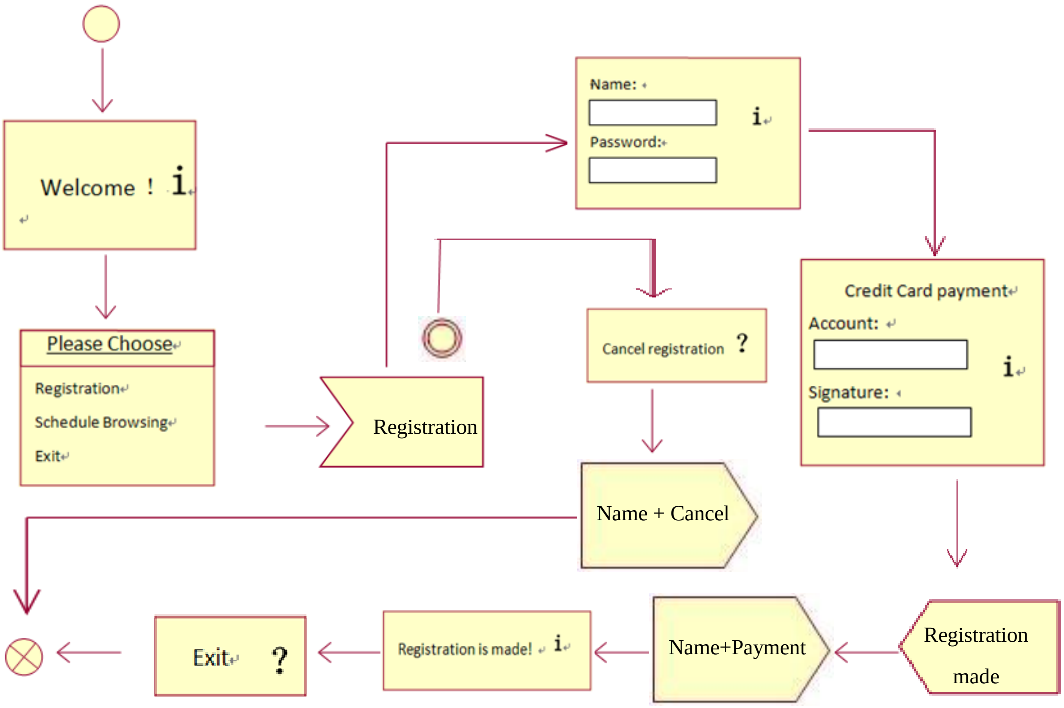


Figure 9. 2 Behaviour Scenario of *Conference Registration*

- **Conferee**, Behaviour Scenario of *Conference Schedule Browsing*

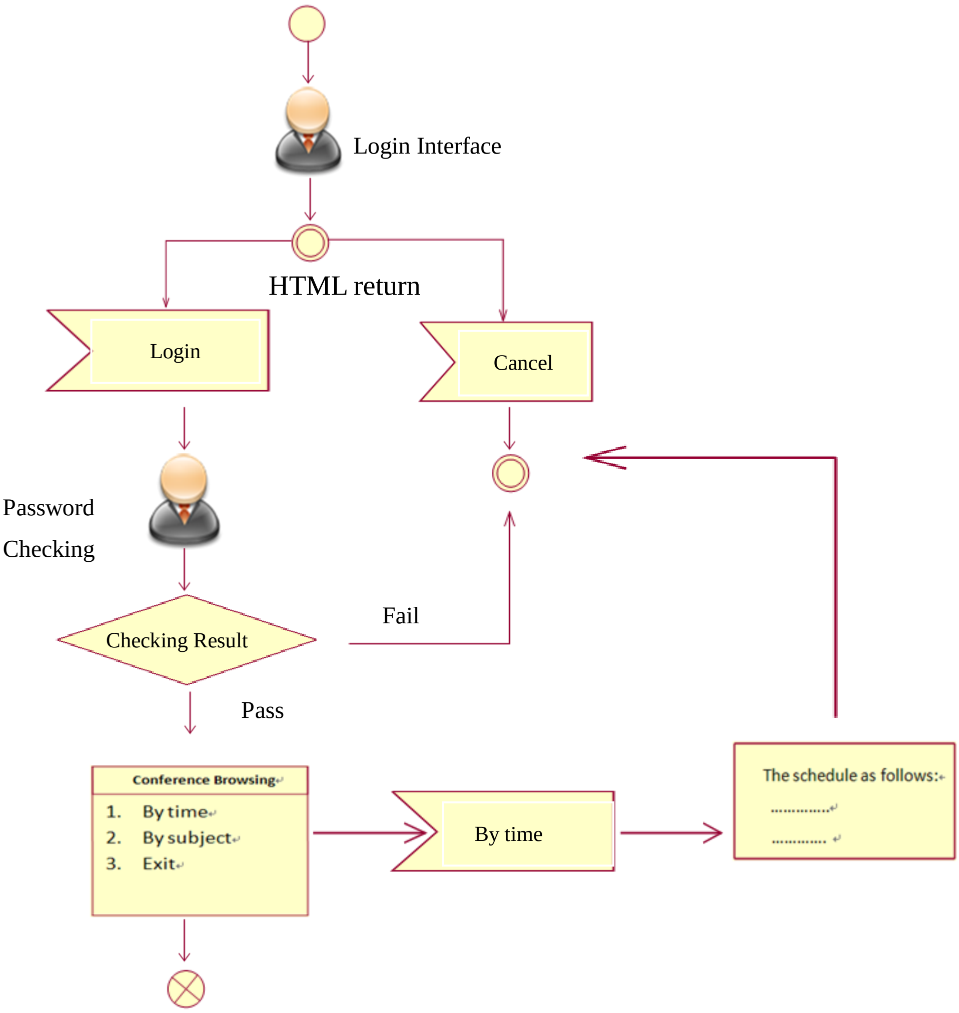


Figure 9. 3 Behaviour Scenario of *Conference Schedule Browsing*

The application model is constructed by using the meta-model of the registration system based on mobile and the primary meta-model of behaviour scenario and the behaviour scenarios of the application model.

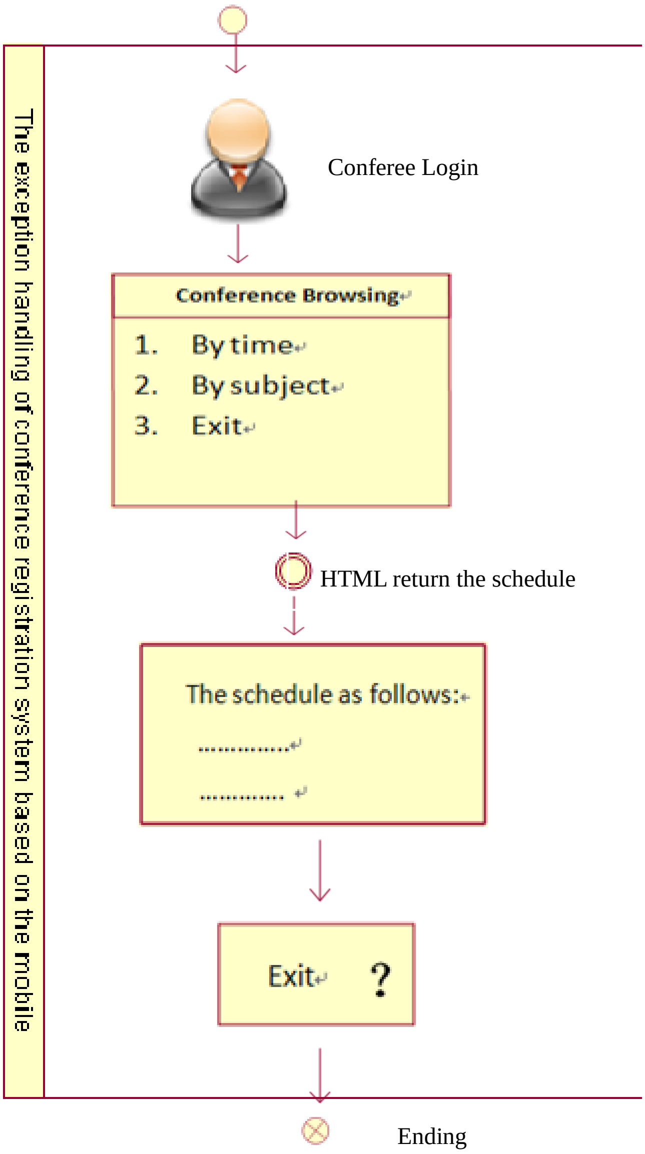


Figure 9. 4 Application Model of Conference Registration System based on mobile

9.3 Online Shopping System

An online shopping system is designed and implemented to illustrate the domain-specific modelling process. It helps to demonstrate the detailed steps for using SODSMI approach to develop domain-specific application system based on the xDSM models.

Nowadays Internet provides us with not only an information platform but also a business trading platform. People can carry out a transaction at home no matter the transaction is B2B (Business to Business), B2C (Business to Consumer) or C2C (Consumer to Consumer). It increases the transaction speed and reduces the transaction cost significantly. The case studies the core fragments of the online shopping system to complete domain analysis and construct the executable domain-specific models.

9.3.1 Domain Analysis

We use the core fragments of the online shopping system to do domain analysis. The business process of the online shopping system involves customer login, to browse classified commodity information, to select commodities and make the order, to select the delivery method, and to pay via online bank system. For most online shopping systems, the business processes are same and the requirements are also common. It can be implemented with browser/server architecture. Certainly, there are different commodity information, customer information and business rules, etc. in the different concrete online shopping systems. The main domain concepts are extracted and analysed as follows:

- Customer: As the main part of online shopping system, Customer contains all information of registered users. Customer logins system via password validation. All procedures of online shopping system are associated with Customer, for examples, order, delivery, and online payment, etc.

- Browse Commodity: Customers browse the commodity information lists offered by merchants in the catalog and query the details.

- Collect Commodity: Customers pick out commodities after browsing then create the order.

- Order: Order is created after Customers Collect Commodity. It is a collection of commodities selected by Customers, and also the basic unit of delivery and payment.

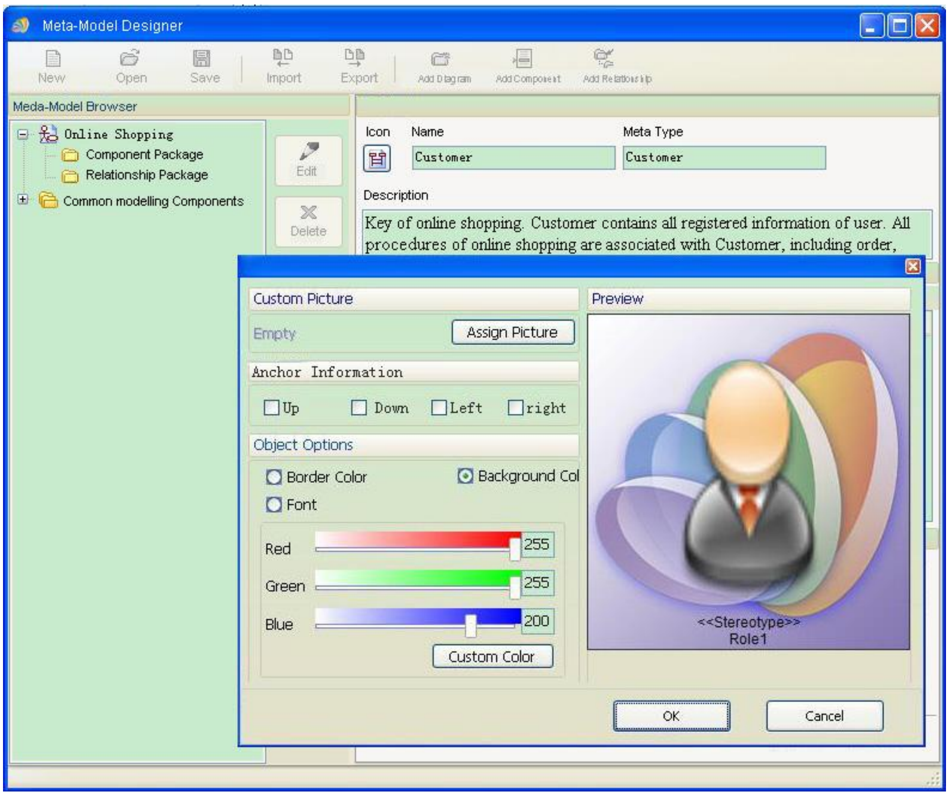
- Commodity Delivery: The commodity delivery information is collected and the delivery cost is calculated according to the order information.

- Select Delivery: Customers select the delivery vendor and the details of the delivery are determined.

- Online Transaction: Customers select the way to pay for the order to complete online transaction, and complete the payment for the transaction by connecting to the online bank.

9.3.2 Meta-Modelling

Based on the above domain analysis, the xDSM meta-model is built within the generic modelling environment of Archware. First of all, it is to establish the domain space of Online Shopping, then to establish the meta-model based on the domain space and visually define the model elements in Meta-Model Designer.



Name: Customer

Meta Type: Customer

Description: Key of online shopping. Customer contains all registered information of user. All procedures of online shopping are associated with Customer, including order,

Custom Picture: Empty

Anchor Information: □ Up □ Down □ Left □ right

Object Options: ○ Border Color ● Background Col ○ Font

Red 255

Green 255

Blue 200

Figure 9. 5 Meta-Model Designer

According to the domain analysis, the relevant domain concepts are extracted to define the xDSM meta-model of online shopping. Firstly, domain entities and their attributes, icons and events of the meta-model are defined in Table 9.2.

Table 9. 2 Meta-Model Entities of Online Shopping System

Icon	Name	Attribute	Event
	Customer	UserID：String UserName：String Registered：Boolean	[DesignTime Event] OnCreate OnClick
	Browse Commodity	BrowseType：int BrowseCommodity：CommodityList	[DesignTime Event] OnCreate OnClick
	Collect Commodity	OrderID：String	[DesignTime Event] OnCreate

	Order	OrderID：String Commodities：CommodityList CommodityCount：int CommodityWeight：Real TotalPrice：Real	[DesignTime Event] OnCreate OnClick
	Commodity Delivery	DeliveryProvider：String Destination：String HandlingFee：Real	[DesignTime Event] OnCreate OnClick
	Select Delivery	DeliveryFinished：Boolean	[DesignTime Event] OnCreate
	Online Transaction	TradeSuccess：Boolean	[DesignTime Event] OnCreate

The primary xDSM meta-model of the domain space of Online Shopping is constructed by defining the domain entities and their attributes, icons and events in the generic modelling environment of Archware. The domain entities are shown in Table 9.2. Events are almost the design time events which are implemented in application modelling by the generic modelling environment of Archware. While model constraints can be attached to the domain entities. They are the same as events defined by AS&MC syntax. They can improve the details of the xDSM meta-model. For examples:

1. Model constraint of “Order” is an invariant: the total price of the commodities in the order equals the value of TotalPrice.

```
Constraint TotalPriceInvariant{
  Declare AllPrice: Real;
AllPrice := 0;
  If (this. Commodities.count>0) then {
        Foreach (Commodity ACommodity in this.Commodities){
        AllPrice := AllPrice+ ACommodity.Price;
}
Return (this. TotalPrice= AllPrice);
} else {
  Return (this. TotalPrice=0);
}
}
```

2. The OnClick event of “Customer” at design time: to initialise and display of the interface of attribute configuration, which is carried out by Archware.

```
    Operation Customer1OnClick{
    SetAttribute(ID, "IsDrawBack", "true" );
    SetAttribute(ID, "BackColor", "80, 0, 0, 255");
      .....
      ShowAttributeInterface;
    }
```

There are three modes to define Operation in Archware. Firstly, to take advantage of other xDSM meta-models to construct Operation based on behaviour scenario. Secondly, Operation is defined by AS&MC syntax. Thirdly, Operation is defined as an abstract operation and implemented by mapping to the specific web service. The xDSM meta-model entities of the domain space of Online Shopping have multiple Operations which carry out system functions contained in domain concepts.

- **Customer**

  - Password Validation [Abstract Operation]

```
Operation  PasswordValidation；Abstract；

InputPin   UserID, Password: string；
```

OutputPin Result: Boolean；

- Login Interface [Abstract Operation]

Operation LoginInterface；Abstract；

InputPin Null；

OutputPin Result:string；

- Login [Behaviour Scenario]

Operation Login；BS；

InputPin Null；

OutputPin Result:string；

➢ **Browse Commodity**

- Browse Commodity [Abstract Operation]

Operation BrowseCommodity；Abstract；

InputPin BrowseType:string；

OutputPin Result:string；

➢ **Collect Commodity**

- Collect Commodity [Behaviour Scenario] [Active Opertion]

Operation CollectCommodity；BS；Active；

InputPin Null；

OutputPin Result:string；

➢ **Order**

- Order Confirmation [Abstract Operation]

Operation OrderConfirm； Abstract；

InputPin OrderID: string；

OutputPin Result:Boolean；

- Order Cancel [Abstract Operation]

Operation OrderCancel； Abstract；

InputPin OrderID: string；

OutputPin Result:Boolean；

- Add Commodity [Abstract Operation]

Operation AddCommodity； Abstract；

InputPin OrderID, CommodityID: string; Num:int；

OutputPin Result:Boolean；

- Delete Commodity [Abstract Operation]

Operation DelCommodity； Abstract；

InputPin OrderID, CommodityID: string; Num:int；

OutputPin Result:Boolean；

➢ **Commodity Delivery**

- Information Collection [Abstract Operation]

Operation DeliveryInfo； Abstract；

InputPin OrderID, DeliveryProvider: string；

OutputPin Result:string；

- Cost Calculation [Abstract Operation]

  Operation CostCalculate； Abstract；

  InputPin OrderID, DeliveryProvider: string；

  OutputPin Result:Real；

- **Select Delivery**

  - Select delivery [Behaviour Scenario] [Active Operation]

    Operation SelectDelivery； BS； Active；

    InputPin OrderID: string；

    OutputPin Result:string；

- **Online Transaction**

  - Trasaction [Behaviour Scenario] [Active Operation]

    Operation Transaction； BS； Active；

    InputPin OrderID: string；

    OutputPin Result:string；

  - Collect Payment Information [Abstract Operation]

    Operation SelectBank； Abstract；

    InputPin Null；

    OutputPin Result:string；

  - Payment Confirmation [Abstract Operation]

    Operation PayConfirm； Abstract；

InputPin OrderID,BankName: string；

OutputPin Result:string；

- Pay Online [Abstract Operation]

Operation PayOnline；Abstract；

InputPin OrderID, BankName: string; Payment: Real；

OutputPin Result:Boolean；

All abstract operations are extracted at the stage of meta-modelling of the domain space of Online Shopping by developers, and corresponded to web services which implement those abstract operations so as to construct AGOS service information of the domain space. For example, Table 9.3 shows the abstract operation of *PasswordValidation* of the modelling entity of **Customer** as follows.

Table 9. 3 Service Information of the Abstract Operation of *PasswordValidation*

ModelID		Meta-OnlineShopping
Abstract Operation	Name	PasswordValidation
	InputPin	UserID,Password: string
	OutputPin	Result: Boolean
ServiceCount		1
Service0	Name	MetaOS- PasswordValidation
	URL	http://202.203.208.127/SOAP/Meta-OnlineShopping
	Protocol	HTTP
	SOAP	<message name=" MetaOS-PVRequest "> <part name=" UserID" type="xs:string"/>

		<part name=" Password" type="xs:string"/> </message> <message name=" MetaOS-PVResponse "> <part name="return" type="xs:boolean "/> </message> <operation name=" MetaOS-PasswordValidation "> <soap:operation soapAction= "urn:Hanks-MetaOS" style="rpc"/> <input message="tns:MetaOS-PVRequest"> <soap:body use="encoded" encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:MetaOS #PV/> </input> <output message="tns: MetaOS-PVResponse"> <soap:body use="encoded" encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/" namespace="urn: MetaOS #PV/> </output> </operation>
	WSDL	http://202.203.208.127/WSDL/Meta-OnlineShopping
	InputMap	ServiceInput. UserID := OPInputPin. UserID; ServiceInput. Password:= OPInputPin.

		Password;
	OutputMap	OPOutputPin.Result := ServiceOutput.return;

The domain space of Online Shopping adopts the extension mechanism of xDSM meta-model. Namely, the existing meta-models of Online Shopping and the primary meta-model of behaviour scenario are collected to define the operations of the meta-model entities of the domain of Online Shopping. They will be fixed into the xDSM meta-model. There are multiple xDSM meta-model entities involving the operations described with behaviour scenarios in the domain space of Online Shopping.

- **Customer**, Behaviour Scenario of *Login* Operation:

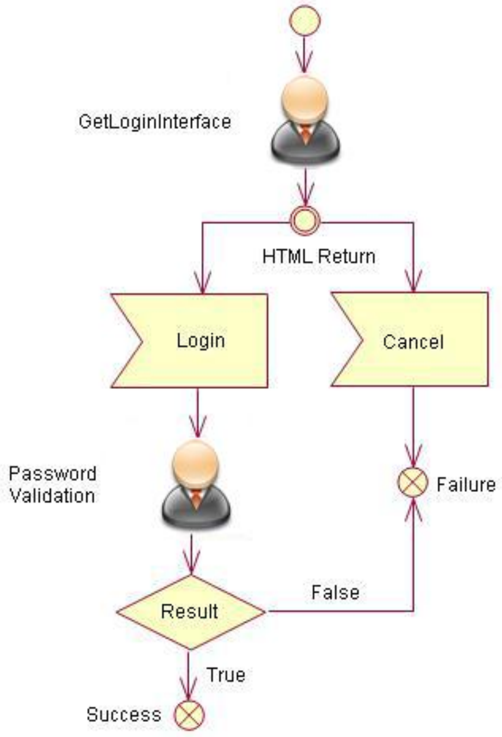


Figure 9. 6 Behaviour Scenario of *Login* Operation

- **Select Delivery**, Behaviour Scenario of *SelectDelivery* Active Operation:

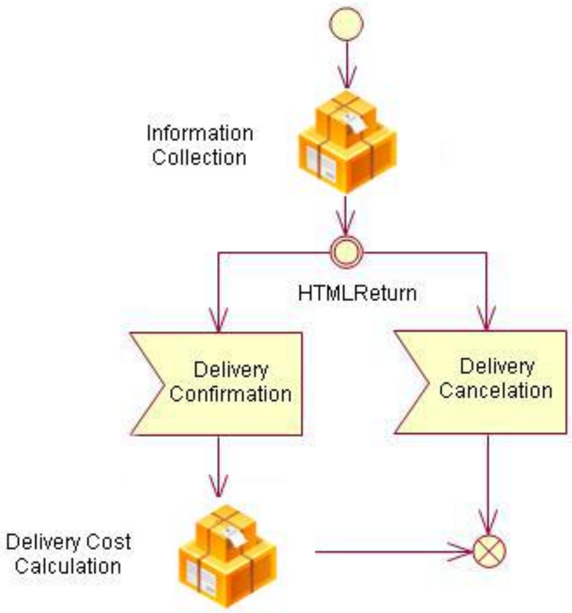


Figure 9. 7 Behaviour Scenario of *SelectDelivery* Active Operation y

- ➢ **Collect Commodity**, Behaviour Scenario of *CollectCommodity* Active Operation:

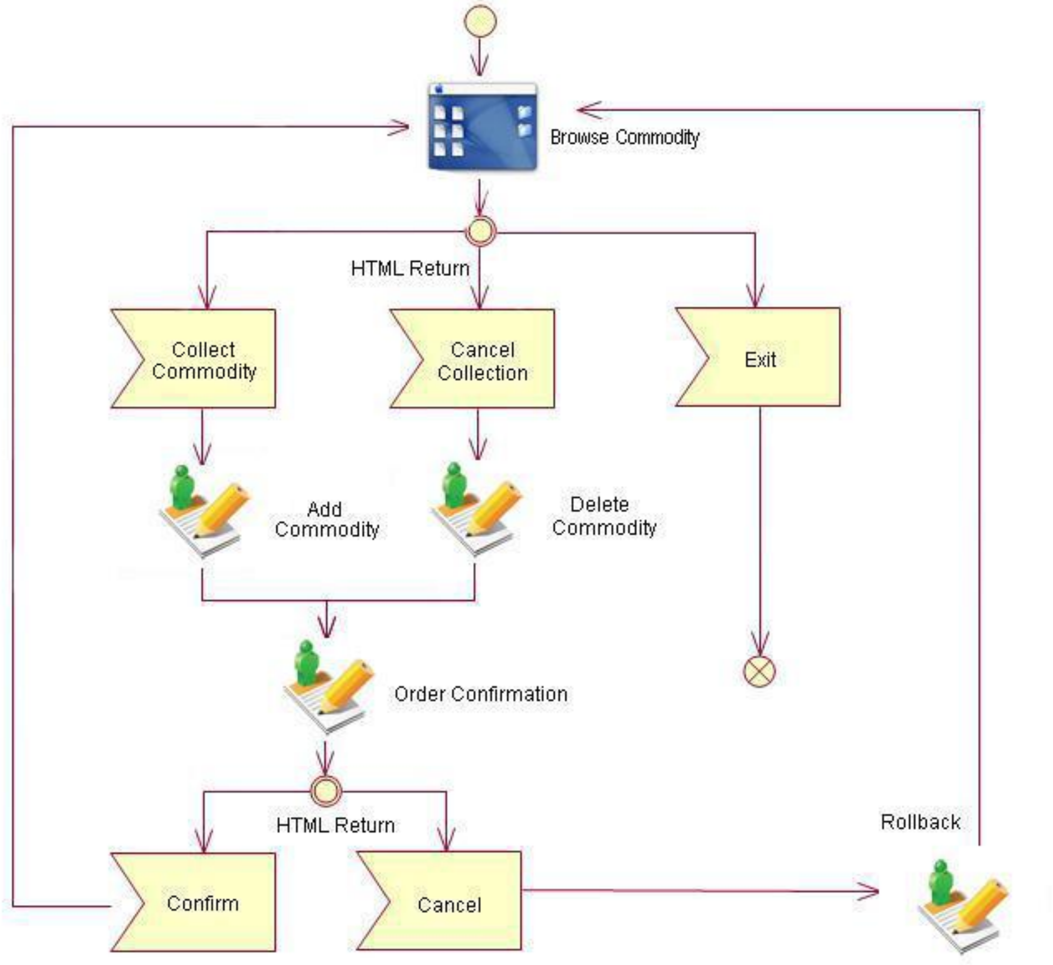


Figure 9. 8 Behaviour Scenario of *CollectCommodity* Active Operation

➢ **Online Transaction**, Behaviour Scenario of *Trasaction* Active Operation:

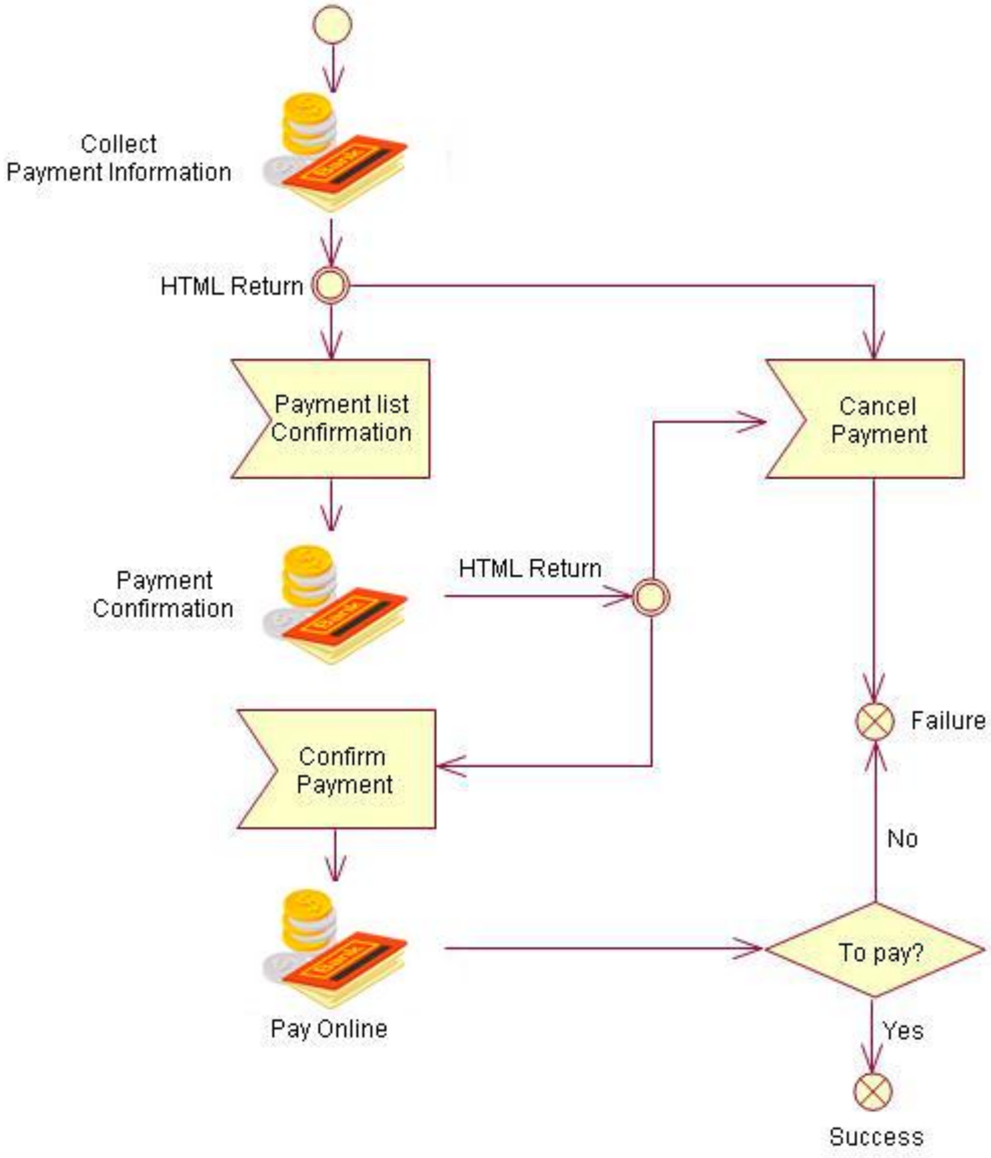


Figure 9. 9 Behaviour Scenario of *Transaction* Active Operation

On one hand, behaviour scenario is used to construct Operations of xDSM model to describe system behaviour. On the other hand, behaviour scenario can refer the meta-models of other domain spaces while it is used to describe Operations so as to support the reuse of domain knowledge and its implementation. The control flow of behaviour scenario is established by using Relationships of the primary meta-model of behaviour scenario. The data flow is established mainly by action specifications attached to Relationship to carry out the association. The action specification is represented as an active operation described by AS&MC syntax. For instance, the data of Login and PasswordValidation are connected by the active operation of the associated relationship between Login and PasswordValidation.

```
Operation SR_ActiveOP1{
      Declare G_Password: string;
      Declare G_UserID: string;
G_Password := ParseXML('Root.Result.Password',
      ComfirmEvent.Message.OutputPin.Result);
G_UserID := ParseXML('Root.Result.UserID',
      ComfirmEvent.Message.OutputPin.Result);
      If (G_UserID<>'' and G_Password<>'') then {
            Customer1. PasswordValidation.InputPin.UserID := G_UserID;
            Customer1. PasswordValidation.InputPin.Password := G_Password;
} else {
      ThrowException('1002','Input Customer LoginInfo Errror');
}
}
```

The sequential relationship associated element can bind the active operation of SR_ActiveOP1 and fix it into the relationship element with the specific role of the meta-model in order to be easier for application modelling.

9.3.3 Application Modelling

Based on the meta-model of the domain space of Online Shopping, requirement analysis is carried out for a concrete online shopping system. The application model is constructed with Archware by using the meta-model of Online Shopping and the primary meta-model of behaviour scenario on the basis of system requirement specification.

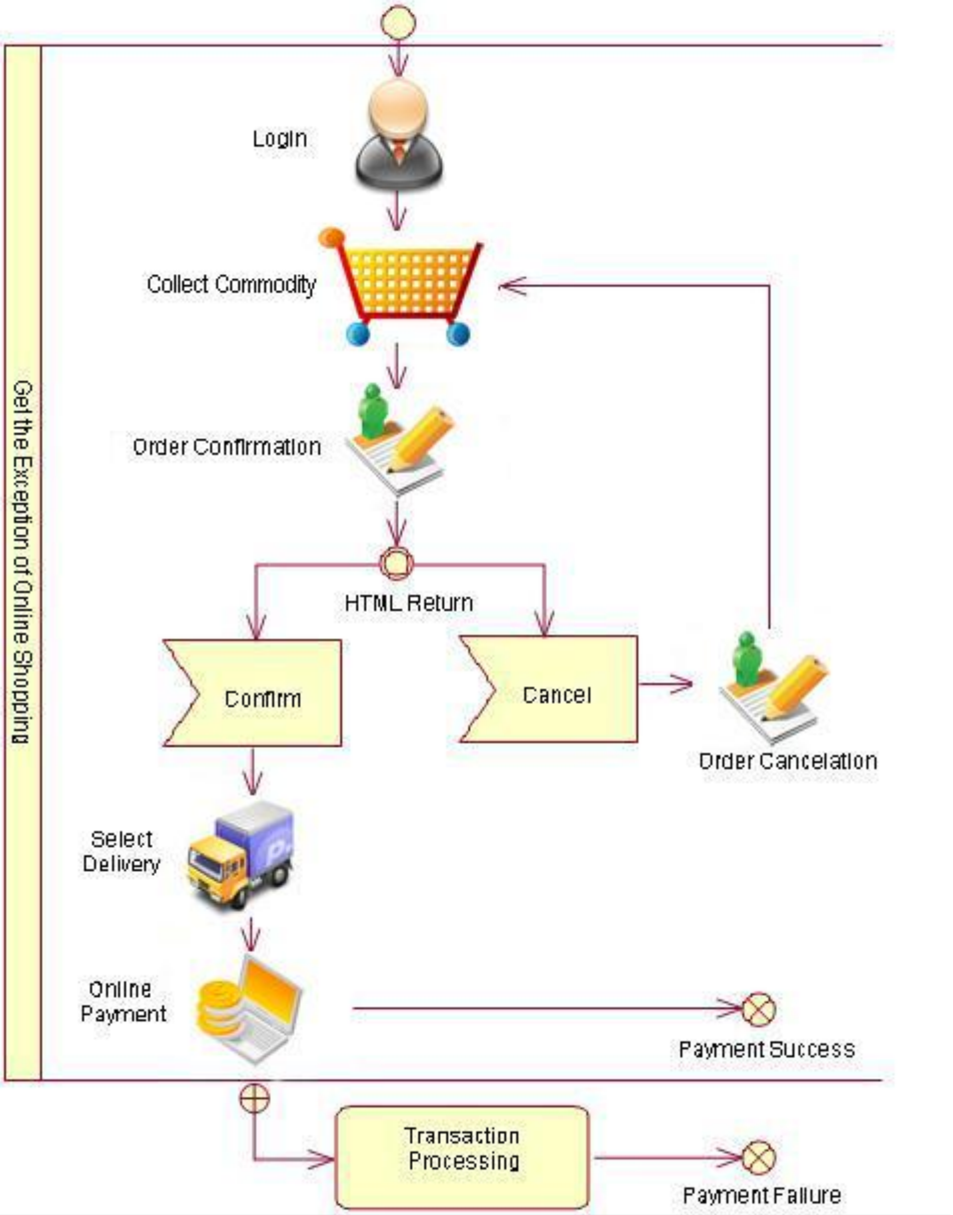
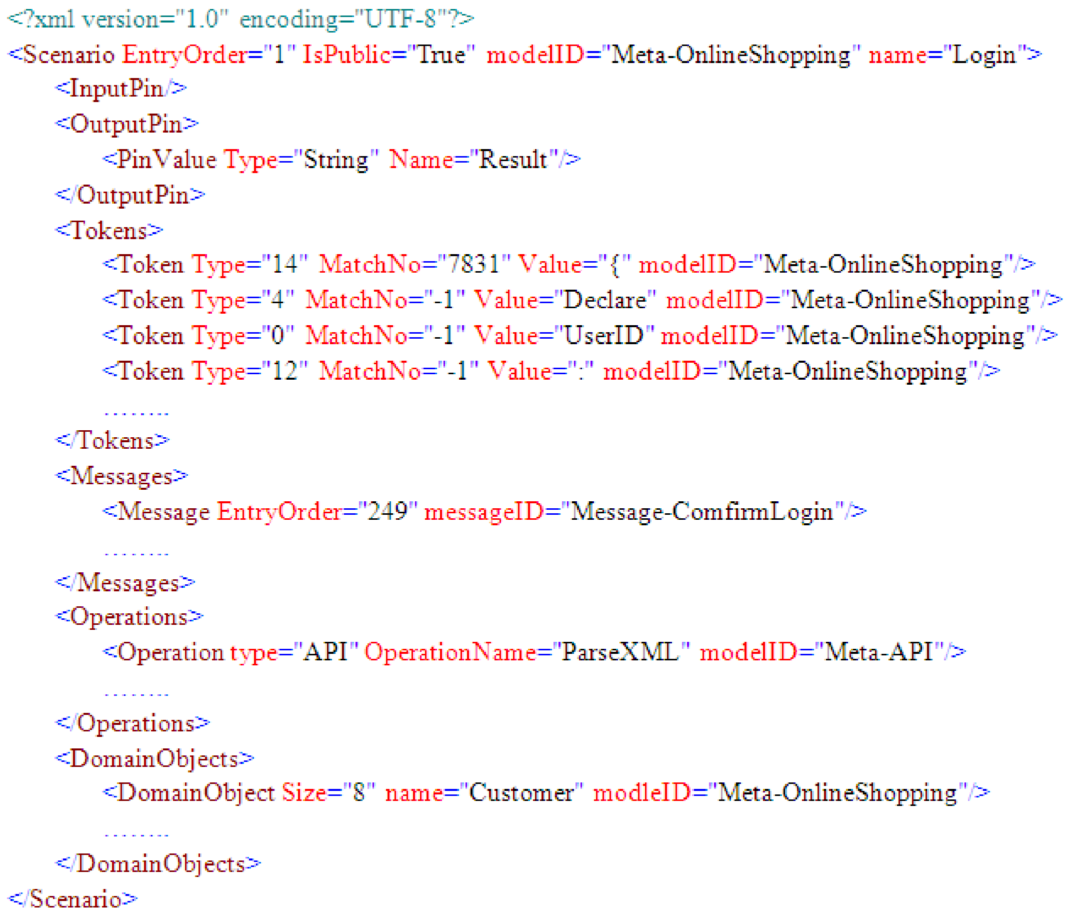


Figure 9. 10 The Application Model of Online Shopping System

Modellers can not only build the xDSM application model but also modify behaviour scenarios which describe operations of the meta-model with Archware, for examples, behaviour scenario of Login and behaviour scenario of CollectCommodity. So the controllability of xDSM model can be improved observably and the application model can be simplified.

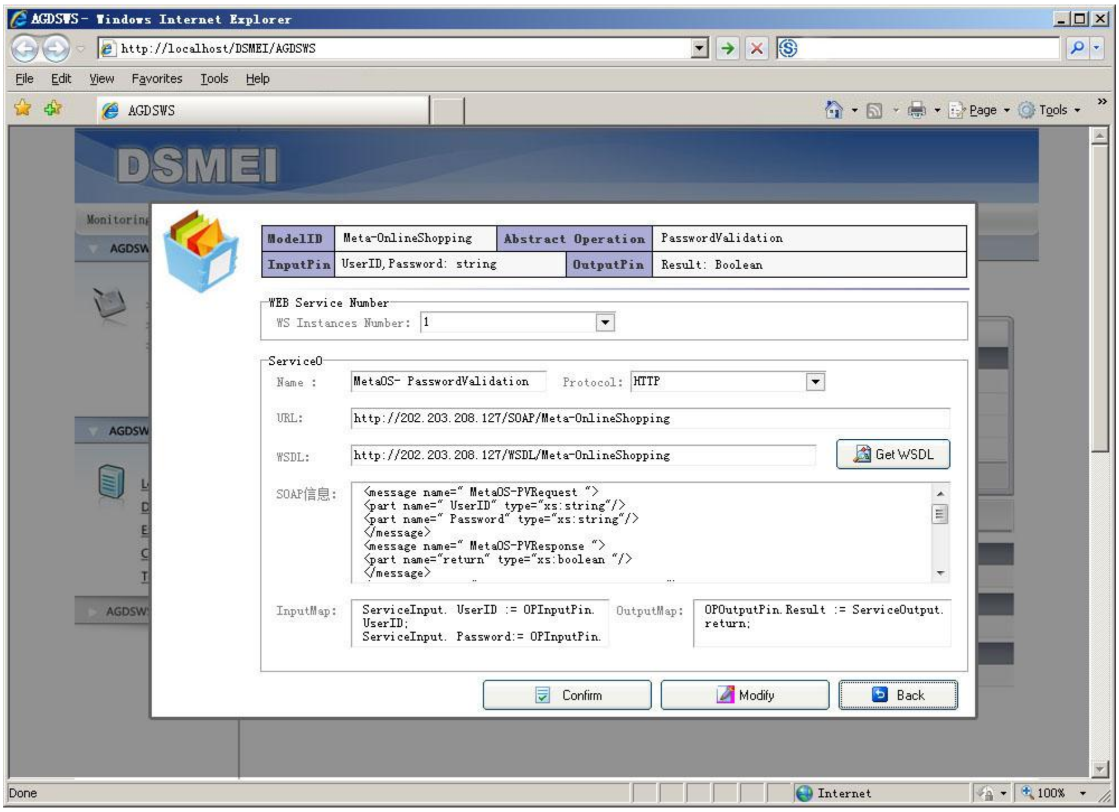
9.3.4 System Implementation

After the completion of domain-specific modelling of the online shopping system, the domain space and the xDSM application model are created. They are the foundation of model execution. DSMEI is the major component of the domain-specific implementation framework. The xDSM application model is loaded and compiled in DSMEI, and transformed into the intermediate code which contains the behaviour logic process and the interface information of xDSM behaviour scenarios. Then the intermediate code is loaded and executed directly by BLEU. For example, the behaviour scenario of Login, the compiled intermediate code is briefly shown as follows:



```
<?xml version="1.0" encoding="UTF-8"?>
<Scenario EntryOrder="1" IsPublic="True" modelID="Meta-OnlineShopping" name="Login">
    <InputPin/>
    <OutputPin>
        <PinValue Type="String" Name="Result"/>
    </OutputPin>
    <Tokens>
        <Token Type="14" MatchNo="7831" Value="{" modelID="Meta-OnlineShopping"/>
        <Token Type="4" MatchNo="-1" Value="Declare" modelID="Meta-OnlineShopping"/>
        <Token Type="0" MatchNo="-1" Value="UserID" modelID="Meta-OnlineShopping"/>
        <Token Type="12" MatchNo="-1" Value=":" modelID="Meta-OnlineShopping"/>
        ........
    </Tokens>
    <Messages>
        <Message EntryOrder="249" messageID="Message-ComfirmLogin"/>
        ........
    </Messages>
    <Operations>
        <Operation type="API" OperationName="ParseXML" modelID="Meta-API"/>
        ........
    </Operations>
    <DomainObjects>
        <DomainObject Size="8" name="Customer" modleID="Meta-OnlineShopping"/>
        ........
    </DomainObjects>
</Scenario>
```

The xDSM application model execution of online shopping system needs the support of AGOS. Service configuration tool of AGOS loads AGOS service information of the domain space of Online Shopping. It is also able to deploy and support web services at run-time.


AGDSWS - Windows Internet Explorer
http://localhost/DSMEI/AGDSWS
File Edit View Favorites Tools Help
AGDSWS
Page Tools
DSMEI
Get WSDL
Confirm
Modify
Back
Done
Internet
100%

ModelID	Meta-OnlineShopping	Abstract Operation	PasswordValidation
InputPin	UserID, Password: string	OutputPin	Result: Boolean

WEB Service Number
WS Instances Number: 1

Service0
Name : MetaOS- PasswordValidation Protocol: HTTP
URL: http://202.203.208.127/SOAP/Meta-OnlineShopping
WSDL: http://202.203.208.127/WSDL/Meta-OnlineShopping
SOAP信息:

```
<message name=" MetaOS-PVRequest ">
<part name=" UserID" type="xs:string"/>
<part name=" Password" type="xs:string"/>
</message>
<message name=" MetaOS-PVResponse ">
<part name="return" type="xs:boolean "/>
</message>
```

InputMap:

```
ServiceInput. UserID := OPInputPin.
UserID;
ServiceInput. Password:= OPInputPin.
```

OutputMap:

```
OPOutputPin.Result := ServiceOutput.
return;
```

Figure 9. 11 Sevices Configuration of AGOS

The xDSM application model of online shopping system is loaded into DSMEI while AGOS service information of the domain space of Online Shopping is loaded into the AGOS service configuration tool. End users visit the guide page of online shopping system via web browser, and transparently call web services provided by DSMEI. The system interfaces are shown as follows:

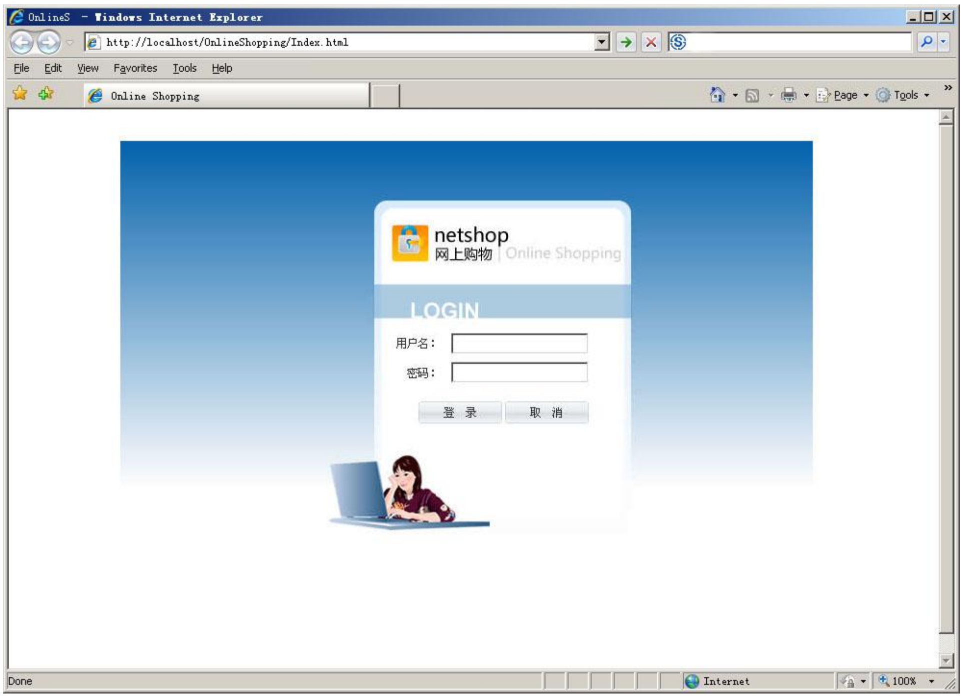
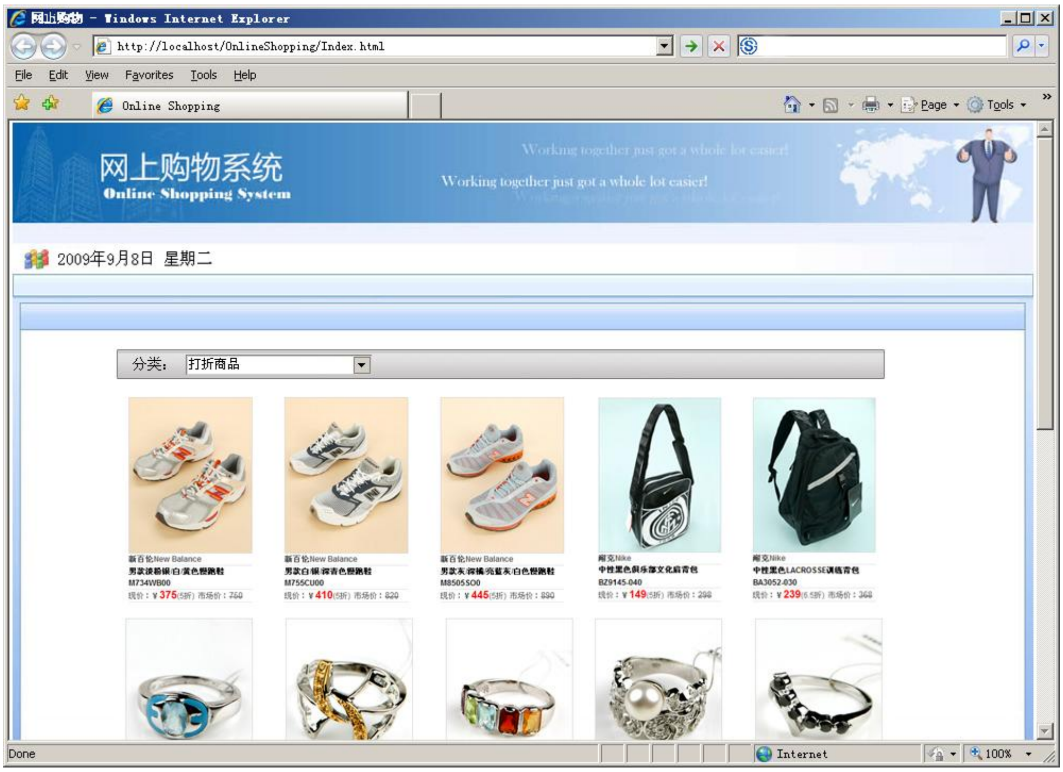

OnlineS - Windows Internet Explorer
http://localhost/OnlineShopping/Index.html
File Edit View Favorites Tools Help
Online Shopping
Page Tools
netshop
网上购物 Online Shopping
LOGIN
用户名：
密码：
登 录
取 消
Done
Internet
100%

Figure 9. 12 The Login Interface of Online Shopping System


网上购物 - Windows Internet Explorer
http://localhost/OnlineShopping/Index.html
File Edit View Favorites Tools Help
Online Shopping
Page Tools
网上购物系统
Online Shopping System
Working together just got a whole lot easier!
2009年9月8日 星期二
分类：
打折商品
Done
Internet
100%

新百伦New Balance
男款淡粉银/白/黄色慢跑鞋
M734WB00
现价：￥375(5折) 市场价：~~750~~

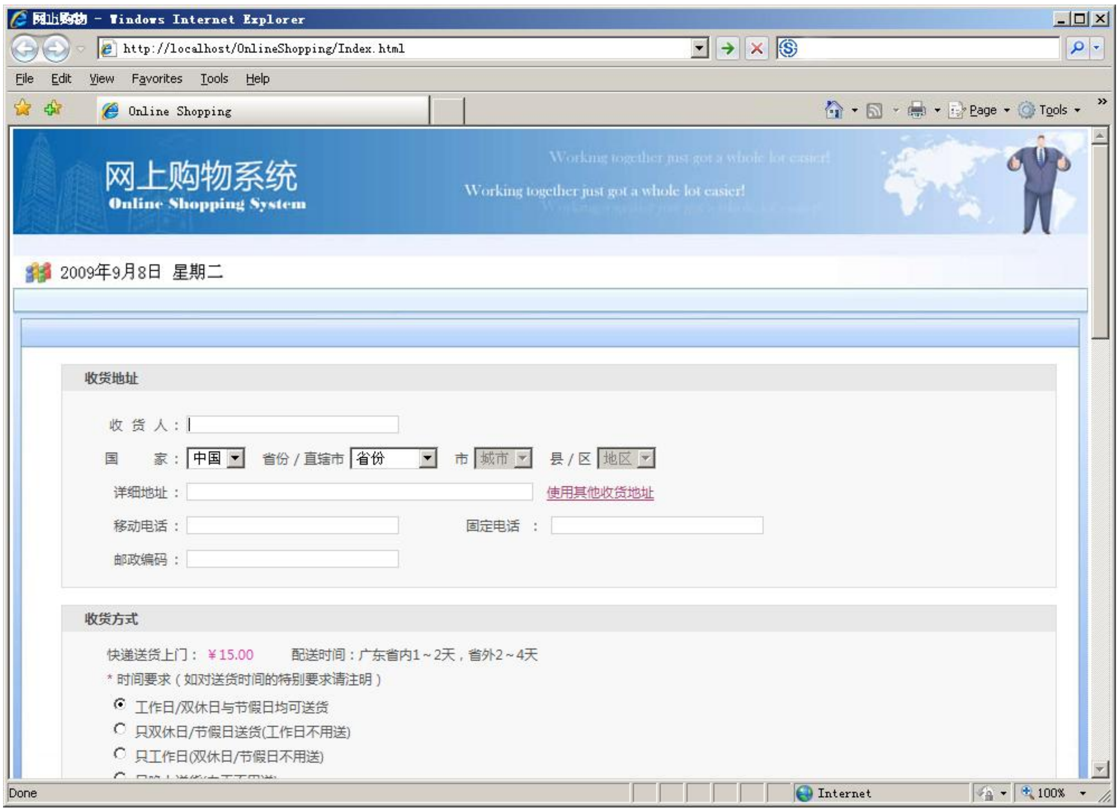
新百伦New Balance
男款白/银/深青色慢跑鞋
M755CU00
现价：￥410(5折) 市场价：~~820~~

新百伦New Balance
男款灰/深橘/亮蓝灰/白色慢跑鞋
M8505SO0
现价：￥445(5折) 市场价：~~890~~

耐克Nike
中性黑色俱乐部文化肩背包
BZ9145-040
现价：￥149(5折) 市场价：~~298~~

耐克Nike
中性黑色LACROSSE训练背包
BA3052-030
现价：￥239(6.5折) 市场价：~~368~~

Figure 9. 13 The Interface of BrowseCommodity


网上购物 - Windows Internet Explorer
http://localhost/OnlineShopping/Index.html
File Edit View Favorites Tools Help
Online Shopping
Page Tools
网上购物系统
Online Shopping System
Working together just got a whole lot easier!
Done
Internet
100%

2009年9月8日 星期二

收货地址

收 货 人：

国　　家：中国　省份 / 直辖市 省份　市 城市　县 / 区 地区

详细地址：　使用其他收货地址

移动电话：　固定电话　：

邮政编码：

收货方式

快递送货上门： ￥15.00　配送时间：广东省内1～2天，省外2～4天

* 时间要求（如对送货时间的特别要求请注明）

● 工作日/双休日与节假日均可送货

○ 只双休日/节假日送货(工作日不用送)

○ 只工作日(双休日/节假日不用送)

Figure 9. 14 The Interface of CommodityDelivery

The domain-specific modelling process and the implementation framework are shown in the case of online shopping system modelling and implementation, involving xDSM meta-modelling of online shopping domain and xDSM application modelling, and executing xDSM application model in DSMEI.

9.4 Summary

In this chapter, two case studies were used to demonstrate that the SODSMI approach can help developers achieve MDD from modelling to system implementation on different domains.

- The case study of conference registration system based on mobile focuses on how to construct the executable domain-specific models including xDSM meta-modelling and xDSM application modelling.

- The case study of online shopping system is designed and implemented to illustrate the domain-specific modelling process. It helps to demonstrate the detailed steps for using SODSMI approach to develop domain-specific application system based on the executable domain-specific model.

Chapter 10

Conclusions and Future Work

MDD is a leap in software development methodology and is the key to solving the "silver bullet" problem. However, models stay at the analysis and design stage over time, and are falling away from system implementation gradually.

The thesis integrates the DSM method and web service techniques with MDD and proposes the SODSMI approach to build the executable domain-specific model and achieve the target of MDD.

In the thesis, xDSM models can be constructed according to MMLs 5 standard to realise MDD. XDML language is designed to construct xDSM models and describe systems integrally, uniformly, detailedly and accurately. Web services are used as software functional entities for xDSM model execution so that the service-oriented domain-specific applications can be implemented by DSMEI without manual intervention.

In the thesis, domain space is proposed to organise domain-specific modelling and implementation. Domain space is the elementary unit of our approach, which can be reused and assembled in order to support the reuse and composition of domain knowledge at architectural level.

10.1 Conclusions

To construct the executable domain-specific models in accordance with MMLs 5 is the target of the thesis. Namely, systems can be integrally, consistently, detailedly and accurately described by models which built with the modelling language. And the service-oriented domain-specific applications can be implemented by DSMEI without manual intervention. The conclusions are drawn as follows:

1) System can be integrally described by xDSM

xDSM is constructed by executable domain-specific modelling based on web services. There are two phases of executable domain-specific modelling: xDSM meta-modelling phase and xDSM application modelling phase. The roles and the responsibilities in the two modelling phases are different. Domain experts/ technical experts, end users complete different work in different phases, and they work together to build the gradually integrated xDSM model:

Firstly, xDSM meta-model and domain framework based on web services can be constructed on the basis of domain analysis by domain experts and technical experts, which make the foundation for xDSM application modelling.

Secondly, in the phase of application modelling, application modelling in GME based on xDSM meta-model is accomplished by end users according to the concrete application requirements. And the final xDSM application model can be executed in DSMEI to verify whether the application requirements have been met. End users raise the application requirements, carry out application modelling and utilise the final result of model execution, which ensures the xDSM application model fully meet the requirements from end users.

During the process of xDSM application modelling, if the xDSM application model constructed by end users cannot achieve system objectives fully, the requirements will switch to domain experts and technical experts. They can improve the xDSM meta-model and its domain framework, and transfer them to end users to be reloaded in GME and DSMEI. This is an interactive process which promotes the integrity of xDSM meta-model and xDSM application model.

2) Systems can be consistently described by xDSM.

The framework of SODSMI ensures that models are consistent with system implementation from two aspects. Firstly, xDSM meta-model and AGOS are collaboratively constructed by domain experts and technical experts who are in the same orgnisation for a specific domain. Abstract operations contained in xDSM meta-model are implemented by AGOS so that behaviour semantics expressed by xDSM meta-model are consistent with the behaviour implementation of AGOS.

Secondly, xDSM application model is created by end users according to domain concepts, rules and constraints defined by xDSM meta-model. xDSM application model is accordant with xDSM meta-model. Meanwhile, xDSM application model is interpreted and executed by DSMEI. xDSM application model can be looked as the executable model as well as the execution entity to accomplish the business behaviour logic with the support of AGOS and achieve system objective. So the integrity of xDSM can be realised.

3) Systems can be accurately described by xDSM in details.

XDML language supports the description and construction of xDSM meta-model and xDSM application model. XDML language integrates well-defined behaviour semantics to support domain-specific behaviour modelling. The concrete syntax of action specifications and model constraints are built on the basis of behaviour semantics of XDML language, which is used to define behaviour details and behaviour constraints of xDSM meta-model and application model, so as to describe systems in detail and accurately. The accurate is limited at the architectural level, not the absolute accuracy.

4) Systems can be implemented without manual intervention.

The most work of creating the executable models is carried out in xDSM meta-modelling phase. Domain space is the elementary unit of the domain-specific modelling and implementation framework. Based on xDSM meta-model, domain space integrates service information of AGOS. Domain space can be loaded into GME and DSMEI, initialising the services configuration of AGOS. After accomplishing xDSM application modelling, xDSM application model can be automatically parsed and executed into the service-oriented domain-specific application to achieve system implementation. Therefore, the system can be realised without manual intervention.

10.2 Success Criteria Revisited

The methodology is proposed in the thesis for architecture-centric domain-specific modelling and implementation for domain-specific software development and reuse, which links models and system implementation. The successes mainly reflect as follows:

1) The executable model, xDSM is constructed based on domain-specific modelling to achieve MDD.

The accurate and integrated executable domain-specific model, xDSM can be constructed based on the framework of service oriented executable domain-specific modelling and implementation. Compared to the traditional modelling methods, the process of xDSM modelling can be divided into two phases, xDSM meta-modelling phase and xDSM application modelling phase.

xDSM meta-model and the corresponding AGOS services are accomplished by domain experts and technical experts in the phase of xDSM meta-modelling based on domain analysis. Not only xDSM meta-model but also the relevant domain framework based on web services are constructed by xDSM meta-modelling. It intensively completes the most work of the executable modelling and reduces the complexity of xDSM application modelling significantly. The reusability of xDSM meta-model and AGOS services confirm that all the work is worthy. xDSM meta-modelling is the foundation of constructing the executable models, and makes it possible that xDSM application model can be transformed directly into domain-specific application system.

In the phase of xDSM application modelling, xDSM meta-model is used by end users who are familiar with the concrete requirements to construct xDSM application model in GME. xDSM application modelling should be relatively simple and intuitive. End users are familiar with the domain concepts which will be used to construct application models. xDSM application model can be directly executed in DSMEI with the support of AGOS.

The xDSM modelling process is an iterative process. xDSM application model is constructed based on xDSM meta-model, while xDSM application model can reflect to xDSM meta-modelling so as to make xDSM meta-model and its corresponded AGOS incrementally improved.

2) The executable model, xDSM is described by XDML language.

XDML language is defined for executable domain-specific modelling. It supports the description and construction of both xDSM meta-model and xDSM application model. XDML language supports the description and construction of xDSM meta-model and xDSM application model. XDML language integrates well-defined behaviour semantics to support domain-specific behaviour modelling. The concrete syntax of action specifications and model constraints are built on the basis of behaviour semantics of XDML language, which is used to define behaviour details and behaviour constraints of xDSM meta-model and application model, so as to describe systems in detail and accurately. XDML language is the foundation for model execution.

3) DSMEI is constructed to realise the direct execution of xDSM models.

The xDSM model cannot be executed directly. It depends on the execution environment to be interpreted and executed. DSMEI is designed and instantiated in the thesis, which includes BLEF, DSPROF and AGOSOF. DSMEI provides execution environment for xDSM application model which is parsed into operation sequences with the accurate semantics, and executes operations to implement the application system. DSMEI integrates domain framework and combines AGOS to provide virtual operations with software functional entities. Therefore, xDSM models become the executable software products and can be executed directly in DSMEI.

4) xDSM application model can be transformed into service-oriented domain-specific application with the support of DSMEI.

The external framework of DSMEI, AGOSOF and DSPROF, are on the basis of web services. As the standard and generic software components, web services provide end users with open and standard application interfaces of xDSM model execution. Meanwhile, web services can be served as software assets for large-scale reuse and provided for xDSM model execution as software functional entities.

Web services model based on business document exchange is proposed. On one hand, the dynamic publishing and calling of domain application web services are realised; on the other hand, the virtualisation of AGOS services is realised. It supports xDSM model execution effectively, and achieves the transformation from xDSM application model to the service-oriented domain-specific application.

5) Domain specific software reuse and composition are achieved via domain

spaces reuse and composition at architectural level so as to realise the reuse and composition of domain knowledge.

Domain space is the basic unit of domain-specific modelling process of implementation framework. Domain space integrates domain framework on the basis of xDSM meta-model, which is the synthetical representation of domain-specific knowledge and its implementation. The reuse and composition of domain spaces are realised by the flexible architecture of domain space on the framework of service oriented executable domain-specific modelling and implementation. It makes software reuse at the domain level, realises the reuse of domain knowledge, and openly extends the range and scale of domain-specific model and its implementation.

10.3 Future Work

The thesis integrates domain-specific modelling and web service techniques with model-driven development and proposes SODSMI approach to build the executable domain-specific model and to achieve the target of model-driven development. But there are still many aspects for improvement and implementation. The further works are as follows:

1) Verification Tools

In the thesis, XDML language is used to define behaviour detail and behaviour constraints of xDSM meta-model and xDSM application model. So the system can be described accurately and in detail. Domain experts/technical experts and end users accomplish different work in different phases to construct incrementally improved xDSM models. Next, the corresponding verification tools will be developed and loaded into DSMEI to ensure the integrity and accuracy of models.

2) Intelligentised Model Execution Infrastructure

DSMEI is designed on the basis of the accurate and integrated description of xDSM models. The model execution infrastructure is relatively simple. We will intelligentise the model execution infrastructure and introduce the intelligentised model parsing and executing mechanism so as to simplify the modelling process.

3) Application and practices

It is necessary for us to utilise the framework of service oriented executable domain-specific modelling and implementation to carry out application practices. The more practical results can help us to improve the approach and to achieve MDD practically.

References

[1] T. Andrews, F. Curbera and H. Dholakia, *Business Process Execution Language for Web Lervices Version 1.1*, IBM, USA, 2003

[2] G. Arango, “Domain Analysis Methods”, *Software Reusability,* Ellis Horwood, New York, 1993, pp.17-49.

[3] B. Barns, and T. Bollinger, “Making reuse cost-effective”, *IEEE Software*, 8(1), pp. 13-24, Jan. 1991.

[4] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice SEI Series*, Addison-Wesley, Chapter 12, 1998.

[5] BEA, IBM, Micosoft, SAP and Siebel, *Business Process Execution Language for Web services*, Packt Publishing, Birmingham, 2003.

[6] BEA Systems, Intalio, SAP and Sun Microsystems, *Web Service Choreography Interface (WSCI) 1.0*, 2002.

[7] K. Beck and C. Andres, *Extreme Programming Explained: Embracechange, 2nd edion*, Addison-Wesley, Massachusetts, 2004.

[8] B. Benatallah, M. Dumas, M-C Fauvet and F. A. Rabhi, “Towards Patterns of Web services Compostion”, *Patterns and Skeletons for Parallel and Distributed Computing*, Springer-Verlag London, 2003, pp. 265 - 296.

[9] K. Bennett, J. Denier, and J. Estublier, "Environments for Software

Maintenance," *Technical Report*, Durham University, UK, 1989.

[10] B. Boehm, A. Egyed, J. Kwan and R. Madachy, "Developing Multimedia Applications with the WinWin Spiral Model," *Proceedings of ESEC/FSE 97 and ACM Software Engineering Notes*, November 1997.

[11] C. Britton, *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*, Addison-Wesley Professional, Massachusetts, 2003.

[12] F. P. Brooks, *No Silver Bullet - Essence and Accidents of Software Engineering*, IEEE Computer, Vol.20, No.4, April 1987, pp. 10-19.

[13] R. Buhr, R. Casselman, "Architectures With Pictures", *Proceedings of OOPSLA '92*, pp.466-483, 1992.

[14] J. Carey, "Prototyping: Alternative Systems Development Methodology," *Information and Software Technology*, Vol. 32, No. 2, 1990.

[15] F. Casati, M. Sayal and M. C. Shan, "Developing e-services for Composing Eservices", *Proceeding of 13th International conference on Advanced Information Systems Engineering(CAiSE)*, Springer Verlag, Berlin, 2001.

[16] D. Champeaux, D. Lea, and P. Faure, *Object Oriented System Development*, Addison-Wesley, Reading Mass, 1993.

[17] T. Clark, A. Evans, P. Sammut and J. Willans, *Applied Metamodelling -- A Foundation for Language Driven Development*, Version 0.1, http://albini.xactium.com/content/index.php?option=com_remository&Itemid=28.

[18] S. Cook, G. Jones, S. Kent and A. C. Wills, *Domain Specific Development with*

Visual Studio Domain-Specific Language Tools, Addison-Wesley, Massachusetts, 2007.

[19] K. Czarnecki and U.W. Eisenecker, *Generative Programming*, Addison-Wesley, Massachusetts, 2000.

[20] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, Massachusetts, 2000.

[21] F. S. David, *Model Driven Architecture: Applying MDA to Enterprise Computing (Paperback)*. Wiley, New York, January 2003.

[22] M. D. Davis, R. Sigal and E. J. Weyuker, *Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science*, Academic Press, Inc, 2008.

[23] T. Dean and J. Cordy, "A Syntactic Theory of Sohare Architecture", *IEEE Transactions on Software Engineering*, Vol. 2 I. No. 4, pp. 302-312, April 1995.

[24] M. Dean, G. Schreiber and S. Bechhofer etc., *OWL Web Ontology Language Reference*, W3C Recommendation, 10 February 2004.

[25] J. Ebert and R. Suttenbach, *Meta-CASE in Practice: A CASE for KOGGE*, Springer Berlin, Heidelberg, 2006.

[26] M. E. Fayad and R. E. Johnson, *Domain Specific Application Frameworks: Frameworks*, John Wiley & Sons, Inc., New York, 2004.

[27] M. Feilkas, "How to Represent Models, Languages and Transformations", *Proceedings of 6th OOPSLA Workshop on DomainSpecific Modelling DSM06*, 2006, pp.171-185.

[28] D. Fensel and C. Bussler, “Web service Modelling Framework WSMF”, *Electronic Commerce Research and Application*, 2002, pp. 113-137.

[29] C. Ferris and J. Farrell，“What are Web services”, *Communications of the ACM*, Vol.46, No.6, ACM, New York, 2003, pp.31-35.

[30] A. Fisher, *CASE: Using Software Development Tools*, New York: John Wiley and Sons, 1988.

[31] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure (Chapter 14, Service Virtualization: Infrastructure and Applications)*, Morgan Kaufmann, San Fransisco, 2004.

[32] M. Fowler, "Language Workbenches: The Killer-App for Domain Specific Languages?", *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, ACM, New York, 2008, pp.224-227.

[33] W. B. Frakes and K. Kang, “Software Reuse Research: Status and Future”, *IEEE Transactions on Software Engineering*, Vol. 31, No. 7. (2005), pp. 529-536, 2005.

[34] K. Frank, “Domain Specific Languages VS. UML”, *Methods and Tools*, Vol.16, No.2, Martinig & Associates, Rue des Marronniers 25, 2008, pp2-8. http://www.codegear.com/tw/downloads.

[35] D. S. Frankel, *Model Driven Architecture:Applying MDA to Enterprise Computing*, John Wiley & Sons, January 2003.

[36] A. Gallo, D. Barkol, R. Vavilala and S. Guthrie, *ASP.NET AJAX in Action*, Manning Publications, Greenwich, CT, 2007.

[37] T. Gardner, C. Griffin, J. Koehler and R. Hauser, *A review of OMG MOF 2.0 Query/View/Transformation Submissions and Recommendations Towards the Final Standard*, IBM White Paper submitted to OMG, September 17, 2003.

[38] D. Garlan and M. Shaw, "An Introduction to Software Architecture: Advances in Software Enpineering and Knowledge Engineering", *World Scientific Publishing Company*, Volume I, 1993.

[39] J. C. Georgas, E. M. Dashofy and R. N. Taylor, "Architecture-Centric Development: A Different Approach to Software Engineering", *ACM Crossroads*, Vol.12, Issue.4, 2006, pp. 6-23.

[40] I Graham, *Object-Oriented Methods: Principles & Practice*, Addison-Wesley Professional, Massachusetts, December 27, 2000.

[41] S. Graham, *Building Web Services with Java-Making Sense of XML, SOAP, WSDL, and UDDI*, China Machine Press, Beijing, 2003.

[42] M. Graphics, *Object Action Language Reference Manual*, http://www.mentor.com/products/sm/techpubs/object-action-language-reference-manual-38098.

[43] J. Greenfield, K. Short, S. Cook and S. Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley, United States, 2004.

[44] M. Griss and R. Kessler, "Building Object-oriented Instrument Kits", *Object Magazine*, 6(2), pp.71-81, April 1996.

[45] R. He and B. Liang, "Using Rule Engine to Replace Code", *Computer World*, 2004-14.

[46] R. Hennicker, H. Hussmann and M. Bidoit, “On the precise meaning of OCL constraints”, *Advances in Object Modelling with OCL*, Springer Berlin, Heidelberg, 2002. pp. 69-84.

[47] D. Hybertson, D. Anh and W. Thomas, “Maintenance of COTS-intensive Software Systems”, *Software Maintenance: Res.and Pract.*, Vol. 9, pp.203-216, 1997.

[48] Ingenieria del Software, *OCL Tools*, http://www.um.es/giisw/ocltools/ocl.htm

[49] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, 1999.

[50] I. Jacobson, M. Griss and P. Jonsson, *Software Reuse -- Architecture, Process and Organization for Business Success*, ACM Press, 1997.

[51] J. H. Johnson, *The CHAOS Report*, The Standish Group International, Inc., 1994.

[52] C. Jones, *Programming Languages Table (PLT2006b)*, Software Productivity Research, 2006.

[53] S. Kelly and J. P. Tolvanen, *Domain Specific Modelling: Enabling Full Code Generation*, Wiley-IEEE Computer Society Press, Los Vaqueros, March 2008.

[54] R. Kieburtz et al., “A Software Engineering Experiment in Software Component Generation”, *Proceedings of 18th International Conference on Software Engineering*, IEEE Computer Society Press, United States, March, 1996.

[55] A. Kleppe, J. Warmer and W. Bast, *MDA Explained: The Model Driven Architecture(TM): Practice and Promise,* Addison-Wesley Professional,

Massachusetts, 2003.

[56] A. Kleppe, J. Warmer and W. Bast, *MDA Explained the Practice and Promise of the Model Driven Architecture*, Addison-Wesley, Massachusetts, February 2004.

[57] D. E. Knuth, “Backus Normal Form VS. Backus Naur Form”, *Communications of the ACM,* Vol.7, No.12, Dec.1964, pp.735-736.

[58] T. Kosar and P. E. M. Lopez, “A Preliminary Study on Various Implementation Approaches of Domain Specific Language”, *Information and Software Technology,* Vol.50, No.5, Butterworth-Heinemann, Newton , 2008, pp.390-405.

[59] H. Kreger, *Web services Conceptual Architecture*, IBM Software Group, 2001. http://www.ibm.com/developerworks/cn/Webservices/ws-wsca/part1/

[60] R. Krikhaar and J. G. Wijnstra, *Architectural Concepts for the Single Product Line*, Philips internal report RWB-508-re-95047, Philips Research, 1995.

[61] T. Kuhne, “What is Model? Language Engineering for Model Driven Software Development”, *Dagstuhl Seminar Proceedings,* 2005.

[62] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle and G. Karsai, “Composing Domain-Specific Design Environments”, *IEEE Computer*, Vol.34, No.11, 2001, pp.44−51.

[63] M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of IEEE (Special Issue on Software Engineering)*, Vol. 19, No. 9, pp. 1060-1076, 1980.

[64] K. Li, Z. Chen, H. Mei and F. Yang, “An Introduction to Domain Engineering”,

Computer Science, Beijing, 1999-5-26, pp.21-25.

[65] M. Little, “Service-Oriented Computing, Transactions and Web services”, *ACM*, New York, Vol.46, No.10, 2003, pp.49-54.

[66] H. Liu, Z. Y. Ma and W. Z. Shao, “Progress of Research on Metamodelling”, *Journal of Software*, Vol.19, No.6, 2008, pp.1317-1327.

[67] F. Liu, Y. Shi, L. Zhang, L. Lin and B. Shi, “Analysis of web services composition and substitution via CCS”, *Data Engineering Issues in E-Commerce and Services*, Vol. 4055/2006, Springer Berlin, Heidelberg, 2006, pp. 236-245.

[68] S. Lohr, *The Programmers who Created the Software Revolution -- Go To*, Basic Books, New York, 2001.

[69] H. Ma, W. Shao and Z. Ma, *Grown UML 2.0*, http://www.uml.org.cn /umlgf/200801073.asp.

[70] S. J. Mellor, A. N. Clark and T. Futagami, “Guest Editors' Introduction: Model Driven Development”, *IEEE Software*, Vol. 20, No. 5, Sep/Oct 2003, pp. 14-18.

[71] S. J. Mellor, *MDA Distilled Principles of Model Driven Architecture*. Addison-Wesley, Massachusetts, 2005.

[72] B Meyer, C Mingins and H Schmidt, "Providing Trusted Components to the Industry", *Computer*, Vol. 31, No. 5, May, 1998, pp. 104-105.

[73] S. Mcharaith, T. C. Son and H. Zeng, “Semantic Web Services”, *IEEE Intelligent Systems*, The IEEE Computer Society, Los Alamitos, Vol.16, No.2, 2001, pp.6-53.

[74] S. Mcharaith and T. C. Son, “Adapting Golog for Composition of Semantic Web services”, *Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers, Toulouse, 2002, pp.482-496.

[75] S. J. Mellor and M. J. Balcer, *Executable UML: A Foundation for Model Driven Architecture,* Addison Wesley, Massachusetts, 2002.

[76] S. J. Mellor, S. Tockey and R. Arthaud, *An Action Language for UML: Proposal for a Precise Execution Semantics,* Springer Berlin, Heidelberg 1999.

[77] MetaEdit Inc., *Domain-Specific Modelling with MetaEdit+ 10 Times Faster than UML,* MetaCase, USA, White Paper, 2005.

[78] G. Miller, “The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information”, *The Psychological Review*, Vol. 63, Issue 2, American Psychological Assn, 1956, pp.81−97.

[79] J. Miller and J. Mukerji, *Model Driven Architecture*, Object Management Group (OMG) document ormsc, July 2001.

[80] J. Miller and J. Mukerji, *MDA Guide Version 1.0.1,* Object Management Group(OMG), Framingham, Massachusetts, June 2003.

[81] National Bureau of Standard, "Guidelines on Software Maintenance," *U.S. Department of Commerce/National Bureau of Standards*, June 1984.

[82] J. Neighbors, “The DRACO approach to constructing software from reusable components”, *IEEE Transactions Software Engineering*, 10(5), pp. 564-574, Sep. 1984.

[83] K. Objecten, *Octopus: OCL Tool for Precise Uml Specifications*, 2005.

http://www.klasse.nl/octopus/index.html

[84] OMG, *Action Semantics for UML,* Object Management Group, 1999.

[85] OMG, *Extensible Markup Language (XML) 1.0*, Object Management Group, Framingham, Massachusetts, 2008.

[86] OMG, *Meta Object Facility Specification, version 1.3*, Object Management Group, Framingham, Massachusetts, June 1999.

[87] OMG, *OCL 2.0 Specification*, Object Management Group, Framingham, Massachusetts, 2005.

[88] OMG, *SOAP Version 1.2*, Object Management Group, Framingham, Massachusetts, 2007.

[89] OMG, *The Common Warehouse Metamodel Specifications*, Object Management Group, Framingham, Massachusetts, 2003.

[90] OMG, *UML 2.0 OCL Specification*, Object Management Group, Framingham, Massachusetts, 2003.

[91] OMG, *Unified Modelling Language Specification, version 2.0*, Object Management Group, Framingham, Massachusetts, 2003.

[92] OMG, *Web services Description Language (WSDL) 1.1*, Object Management Group, Framingham, Massachusetts, 2001.

[93] OMG, *XML Metadata Interchange Specification version1.2*, Object Management Group, Framingham, Massachusetts, 2003.

[94] OMG, *XML Schema Part 1: Structures Second Edition*, Object Management

Group, Framingham, Massachusetts, 2004.

[95] OMG, *xtUML Specification*, Object Management Group, Framingham, Massachusetts, 2005.

[96] Y. Papakonstantinon and V. Vianu, “DTD Inference for Views of XML Data”, *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, New York, 2000, pp: 35 - 46.

[97] Y. Papakonstantinou, H. Garcia-Molina and J. Ullman, "MedMaker: A Mediation System Based System Based on Declarative Specifications", *Proceedings of the 12th International Conference on Data Engineering (ICDE'96),* icde, Louisiana, 1996, pp.132.

[98] M. P. Papazoglou, “Service-Oriented Computing: Concepts, Characteristics and Directions”, *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE'03),* IEEE Computer Society Press, Los Alamitos, CA, USA, 2003, pp.3-12.

[99] C. Peltz, “Web services orchestration and choreography”, *Computer,* Computer Publication, USA, Vol.36, No.10, 2003, pp.46-52.

[100] D. Perry and A. Wolf, “Foundations for the Study of Software Architecture”, *ACM Software Engineering Notes*, 17(7), pp.40-52, 1992.

[101] G. D. Plotkin, “A Structural Approach to Operational Semantics”, *Journal of Logic and Algebraic Programming (JALP,* 60-61), 2004, pp.17-139.

[102] G. D. Plotkin, “The Origins of Structural Operational Semantics”, *Journal of Logic and Algebraic Programming (JALP,* 60-61), 2004, pp.323-351.

[103] J. Poole, “Model Driven Architecture: Vision, Standards and Emerging Technologies”, *Proceedings of the Workshop on Metamodelling and Adaptive Object Models (ECOOP 2001)*, 2001.

[104]R. S. Pressman, *Software Engineering: A Practitioner’s Approach*, Fifth Edition, McGraw Hill, 2001.

[105] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill Publishing Co., CA, 2004.

[106] R. Prieto-Diaz, “Status Report: Software Reusability”, *IEEE Software*, 10(3), pp. 61-66, May 1993.

[107] R. Prieto-D íaz and G. Arango (eds.), *Domain Analysis and Software Systems Modelling*, IEEE Computer Society Press, Los Alamitos, CA, 1991.

[108] C. Raistrick, P. Francis and J. Wright, *Model Driven Architecture with Executable UML*, Cambridge University Press, Cambridge, 2004.

[109] R. K. Runde and K. Stolen, “What Is Model Driven Architecture”, *University of Oslo Department of Informatics*, Research Report 304, September 2003.

[110] R. S. Scowen, *Extended BNF -- A Generic Base Standard*, Software Engineering Standards Symposium, 1993.

[111] A. Shalloway and J. R. Trott, *Design Patterns Explained: A New Perspective on Object-Oriented Desig*n, Addison-Wesley Professional, Massachusetts, 2004.

[112] M. Shaw and D. Garlan, *Software Architecture: Perspectives of an Emerging Discipline*, Preutice-Hd, 1996.

[113] R. Soley and OMG Staff Strategy Group, M*odel Driven Architecture white*

paper Draft 3.2, 2000.

[114] J. Sprinkle and G. Karsai, “A Domain Specific visual Language for Domain Model Evolution”, *Journal of Visual Languages and Computing*, Vol.15, No.2, Nashville, 2004, PP. 291-307.

[115] F. Steimann, *Why Most Domain Models are Aspect Free*, August 2004. http://www.kbs. uni-hannover.de/steimann/published/UML2004AOM.pdf.

[116] R. Studer, S. Grimm and A. Abecker, *Semantic Web Services: Concepts, Technologies, and Applications*, Springer, Heidelberg, 2007.

[117] Q. Sun, Y. L. Cao and Z. H. Zhang, “Next Generation Technology of UML Execution and Translation -- ~X_TUML”, *Computer Engineering*, Issue.8,2004, pp.90-91.

[118] T. Tamai and Y. Torimitsu, “Software lifetime and Its Evolution Process over Generations”, *Proceedings of IEEE Conference on Software Maintenance*, Orlando, FL, pp. 63-69, 1992.

[119] UDDI Org, *Universal description, discovery, and integration (UDDI)*. http://www.uddi.org

[120] M. Vӧlter and J. Bettin, *Patterns for Model Driven Software Development*, Version 1.4, May 10, 2004.

[121] W3C Working Group, *Web service architecture*, http://www.w3.org/tr/ws-arch/. 2003.

[122] Y. Wand, “Ontology as a foundation for meta-modelling and method engineering”, *Journal of Information and Software Technology*, Vol.38, No.4,

Springer, Berlin, 1996, pp. 281–288.

[123] J. Warmer and A. Kleppe, *The object constraint language*, Addison-Wesley, 2002.

[124] J. Warmer and A. Kleppe, *Object Constraint Language: Getting Your Models Ready for MDA, Second Edition*, Addison-Wesley Professional, Massachusetts, 2003.

[125] K. Whitehead, *Component-Based Development Principles and Planning for Business Systems*, Addison-Wesley Professional, Massachusetts, 2003.

[126] I. Wilkie, A. King, M. Clarke, Ch. Weaver, Ch. Raistrick and P. Francis, *UML ASL Reference Guide - ASL Language Level 2.5 - Manual Revision D*, Kennedy Carter, 2003.

[127] N. Wirth, “What can we do about the unnecessary diversity of notation for syntactic definitions?” *Communications of the ACM,* Vol. 20, Issue 11, November 1977, pp. 822–823.

[128] F. Yang, H. Mei, J. Lu and Z. Jin, “Some Discussion on the Development of Software Technology”, *Acta Electronica Sinica,* Vol.26, No.9, 2003, pp.1104-1115.

[129] H. Yang and M. Ward, *Successful Evolution of Software Systems*, Artech House, 2003.

[130] Y. Yu and Y. Gu, “Domain Feature Space Based Semantic Representation of Component”, *Journal of Software*, Vol.13, No.2, 2002. pp.311-316.

[131] W. Zhang, J. Huai and X. Li, “Software specification of goal and operation”,

Proceeding of China National Computer Conference 2003, Tsinghua University Press, Beijing, 2003, pp.1610～1618.

[132] W. Zhang and H. Mei, "A Feature-Oriented Domain Model and Its Modelling Process", *Journal of Software*, Vol.14, No.8, Beijing, 2003-8-14, pp.1345-1356.

[133] H. Zhou, X. P. Sun and Q. Duan etc., "XMML: A Visual Metamodelling Language for Domain Specific Modelling and Its Application in Distributed Systems", *Proceedings of 12th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Kunming, China, October 21-23, 2008, pp. 133-139.

Appendix A　Concrete Syntax of XDML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
        <xs:element name="DSMProject">
        <xs:annotation>
        <xs:documentation>Comment describing your root element
    </xs:documentation>
        </xs:annotation>
        <xs:ComplexType>
        <xs:sequence>
        <xs:element ref="Models" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="id"/>
        <xs:attribute name="name"/>
        <xs:attribute name="version"/>
        </xs:ComplexType>
        </xs:element>
        <xs:ComplexType name="ModelType"/>
        <xs:ComplexType name="ModelsType">
        <xs:sequence>
        <xs:element ref="Model" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        </xs:ComplexType>
        <xs:ComplexType name="EntitiesType">
        <xs:sequence>
```

```
<xs:element ref="Entity" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:ComplexType>
<xs:ComplexType name="SpecificationType">
<xs:sequence>
<xs:element name="SpecsItem" minOccurs="0" maxOccurs="unbounded">
<xs:ComplexType>
<xs:sequence>
<xs:element name="content"/>
</xs:sequence>
<xs:attribute name="id"/>
<xs:attribute name="type"/>
<xs:attribute name="lang"/>
</xs:ComplexType>
</xs:element>
</xs:sequence>
</xs:ComplexType>
<xs:element name="Models" type="ModelsType"/>
<xs:element name="Entities" type="EntitiesType"/>
<xs:ComplexType name="PropertiesType"/>
<xs:element name="Properties">
<xs:ComplexType>
<xs:complexContent>
<xs:extension base="PropertiesType">
<xs:sequence>
<xs:element name="Property" minOccurs="0" maxOccurs="unbounded">
```

```
<xs:ComplexType>
<xs:sequence>
<xs:element ref="Properties"/>
</xs:sequence>
<xs:attribute name="type" use="required"/>
<xs:attribute name="name" use="required"/>
<xs:attribute name="value"/>
</xs:ComplexType>
</xs:element>
</xs:sequence>
<xs:attribute name="propertyMgr"/>
</xs:extension>
</xs:complexContent>
</xs:ComplexType>
</xs:element>
<xs:ComplexType name="EventsType">
<xs:sequence>
<xs:element name="Event" minOccurs="0" maxOccurs="unbounded">
<xs:ComplexType>
<xs:attribute name="type"/>
<xs:attribute name="scriptFile"/>
</xs:ComplexType>
</xs:element>
</xs:sequence>
</xs:ComplexType>
<xs:element name="Events" type="EventsType"/>
<xs:ComplexType name="RelationshipsType">
<xs:sequence>
```

```
          <xs:element name="Relationship" minOccurs="0"
  maxOccurs="unbounded">
          <xs:ComplexType>
          <xs:sequence>
          <xs:element name="Roles">
          <xs:ComplexType>
          <xs:sequence>
          <xs:element name="Role" minOccurs="2" maxOccurs="2">
          <xs:ComplexType>
          <xs:sequence>
          <xs:element ref="Properties"/>
          <xs:element ref="Events"/>
          <xs:element ref="Specification"/>
          </xs:sequence>
          <xs:attribute name="type"/>
          <xs:attribute name="elementId"/>
          </xs:ComplexType>
          </xs:element>
          </xs:sequence>
          </xs:ComplexType>
          </xs:element>
          <xs:element ref="Events"/>
          <xs:element ref="Properties"/>
          <xs:element ref="Specification"/>
          </xs:sequence>
          <xs:attribute name="id"/>
          <xs:attribute name="type"/>
          </xs:ComplexType>
```

```
        </xs:element>
        </xs:sequence>
        </xs:ComplexType>
        <xs:ComplexType name="DiagramsType">
        <xs:sequence>
        <xs:element name="Diagram" minOccurs="0"
maxOccurs="unbounded">
        <xs:ComplexType>
        <xs:complexContent>
        <xs:extension base="DiagramType">
        <xs:sequence>
        <xs:element name="VisualElements">
        <xs:ComplexType>
        <xs:sequence>
        <xs:element ref="VisualElement" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        </xs:ComplexType>
        </xs:element>
        <xs:element ref="Properties"/>
        </xs:sequence>
        <xs:attribute name="id"/>
        <xs:attribute name="type"/>
        <xs:attribute name="RenderEngine"/>
        </xs:extension>
        </xs:complexContent>
        </xs:ComplexType>
    </xs:element>
```

```
</xs:sequence>
</xs:ComplexType>
<xs:ComplexType name="DiagramType"/>
<xs:element name="Relationships" type="RelationshipsType"/>
<xs:element name="Diagrams" type="DiagramsType"/>
<xs:element name="CodeGenerators">
<xs:ComplexType>
<xs:sequence>
<xs:element name="CodeGenerator" minOccurs="0" >
<xs:ComplexType>
<xs:sequence>
<xs:element name="script"/>
</xs:sequence>
<xs:attribute name="id"/>
<xs:attribute name="type"/>
<xs:attribute name="lang"/>
</xs:ComplexType>
</xs:element>
</xs:sequence>
</xs:ComplexType>
</xs:element>
<xs:element name="Specification" type="SpecificationType"/>
<xs:ComplexType name="EntityType">
<xs:sequence>
<xs:element name="RefinedModel">
<xs:ComplexType>
<xs:sequence>
<xs:element ref="Model" minOccurs="0"/>
```

```
</xs:sequence>
</xs:ComplexType>
</xs:element>
<xs:element name="Attachment">
<xs:ComplexType>
<xs:sequence>
<xs:element ref="Entity" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:ComplexType>
</xs:element>
<xs:element name="Contained">
<xs:ComplexType>
<xs:sequence>
<xs:element name="EntityRef" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:ComplexType>
</xs:element>
<xs:element ref="Properties"/>
<xs:element ref="Events"/>
<xs:element ref="Specification"/>
</xs:sequence>
<xs:attribute name="id"/>
<xs:attribute name="type"/>
</xs:ComplexType>
<xs:element name="Entity" type="EntityType"/>
<xs:element name="Model">
<xs:ComplexType>
```

```
        <xs:complexContent>
        <xs:extension base="ModelType">
        <xs:sequence>
        <xs:element ref="Entities"/>
        <xs:element ref="Relationships"/>
        <xs:element ref="Diagrams"/>
        <xs:element ref="Events"/>
        <xs:element ref="Properties"/>
        <xs:element ref="Specification"/>
        <xs:element ref="CodeGenerators"/>
        <xs:element name="RefEntities">
        <xs:ComplexType>
        <xs:sequence>
        <xs:element name="RefEntity" minOccurs="0"
maxOccurs="unbounded">
        <xs:ComplexType>
        <xs:attribute name="id"/>
        <xs:attribute name="ModelId"/>
        </xs:ComplexType>
        </xs:element>
        </xs:sequence>
        </xs:ComplexType>
        </xs:element>
        </xs:sequence>
        <xs:attribute name="id"/>
        <xs:attribute name="type"/>
        </xs:extension>
        </xs:complexContent>
```

```
</xs:ComplexType>
</xs:element>
<xs:ComplexType name="VisualElementType">
<xs:sequence>
<xs:element ref="Div" maxOccurs="unbounded"/>
<xs:element name="Scripts"/>
</xs:sequence>
<xs:attribute name="id"/>
<xs:attribute name="elementId"/>
<xs:attribute name="events"/>
</xs:ComplexType>
<xs:ComplexType name="DivType">
<xs:sequence>
<xs:element ref="Div" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id"/>
<xs:attribute name="style"/>
<xs:attribute name="features"/>
<xs:attribute name="events"/>
</xs:ComplexType>
<xs:element name="Div" type="DivType"/>
<xs:element name="VisualElement">
<xs:ComplexType>
<xs:sequence>
<xs:element ref="Div"/>
<xs:element name="Script">
<xs:ComplexType>
<xs:attribute name="lang"/>
```

```
            </xs:ComplexType>
            </xs:element>
            </xs:sequence>
            <xs:attribute name="id"/>
            <xs:attribute name="type"/>
            <xs:attribute name="elementId"/>
            <xs:attribute name="events"/>
            </xs:ComplexType>
            </xs:element>
    </xs:schema>
```

Appendix B List of Publications

[1] Qing Duan, Zhihong Liang, etc., “Developing Distributed Virtual Machines for the Tri-Integration-Pattern Based Platform (TIPBP)”, *Proceedings of IEEE International Workshop on Service-Oriented System Engineering (SOSE2005),* Beijing, China, October 20-21, 2005.

[2] Qing Duan and Hongji Yang, “An Application Framework for the Tri-Integration Pattern”, *Proceedings of Postgraduate Research Conference in Electronics, Photonics, Communications & Networks, and Computing Science (PREP2005),* Lancaster, UK, April 2005.

[3] Yang Xu, Qing Duan and Hongji Yang, “Business Rules Based web services Oriented Customer Relationship Management System (CRM) Evolution”, *Proceedings of Workshop on Software Technology and Engineering Practice (STEP 2005) at the 21st IEEE International Conference on Software Maintenance (ICSM 2005),* Budapest, Hungary, September 24-25, 2005.

[4] Hua Zhou, Xingping Sun, Qing Duan, etc.,“XMML: A Visual Meta-Modelling Language for Domain-Specific Modelling and its Application in Distributed Systems”, *Proceedings of 12th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS),* Kunming, China, October 21-23, 2008. pp. 133-139.

[5] Huaiyan Gao, Hua Zhou and Qing Duan, “A Reengineering Assistant and Case Studies”, *Scientific Research Monthly Journal*, no.4, Serial 4, December 2004, pp. 38-42.