A DOMAIN-SPECIFIC MODELING APPROACH FOR COMPONENT-BASED SOFTWARE DEVELOPMENT

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

DOCTOR OF EDUCATION

BY

ZHIHUI YANG

DISSERTATION ADVISOR: DR. WAYNE ZAGE

BALL STATE UNIVERSITY

MUNCIE, INDIANA

JULY 2009
Contents

ACKNOWLEDGEMENTS ................................................................................................................. ix
ABSTRACT ........................................................................................................................................ x

1. Introduction

1.1. Motivation ............................................................................................................................... 1

1.2. Contributions .......................................................................................................................... 3

2. Related Work

2.1 Component-based software development ................................................................................ 6

2.2 Domain-Specific Modeling .................................................................................................... 7

2.3 Domain-specific Modeling and Software Component ............................................................ 9

2.4 Dependable Component-Based Software Systems ................................................................. 11

Part I - Domain-Specific Modeling for Component Composition

3. Component Composition

3.1 CBSD Overview ...................................................................................................................... 14

3.2 Approaches of Component Composition ................................................................................ 17

4. Domain-specific modeling
4.1 What is domain-specific modeling ............................................. 20

4.2 Why domain-specific modeling ............................................. 21

4.3 Domain-specific modeling vs. UML ........................................ 22

4.4 Eclipse-Based DSM Framework ........................................... 23
  4.4.1 Eclipse Overview ......................................................... 23
  4.4.2 Eclipse Architecture .................................................... 25
  4.4.3 Eclipse Modeling Framework ......................................... 27
  4.4.4 Eclipse Graphical Modeling Framework ............................. 29

5. Model-based component composition environment for a specific domain

  5.1 Overview ........................................................................ 33

  5.2 Domain Meta-model ........................................................ 34

    5.2.1 Modeling Component .................................................. 37

    5.2.2 Modeling Component Association .................................. 38

    5.2.3 Modeling Component Aggregation and Inheritance ........... 42

  5.3 Model Instance .............................................................. 45

  5.4 Code generation .............................................................. 47
5.4.1 Model Code Generation ............................................. 48
5.4.2 Editor Code Generation ............................................. 52

6. Mobile Service Creation Framework (MSCF)

6.1 Mobile Service Overview ............................................. 56
6.2 Mobile Operating Systems ............................................. 60
6.3 Messaging-based Mobile Services ..................................... 64
6.4 High-level Architecture of Messaging-based Mobile Services .................................................. 68
6.5 Component-based Framework for Mobile Service Creation

6.5.1 Related Work ......................................................... 71
6.5.2 Mobile Service Creation Framework (MSCF) ..................... 73
6.5.3 Domain Meta-model of MSCF ....................................... 78
6.5.4 Elements of MSCF Domain Meta-model .......................... 83
6.5.5 MSCF Visual Editor ................................................ 103

6.6 MSCF Case Study – Voting Service

6.6.1 Overview .......................................................... 110
6.6.2 Development Usage Scenario ........................................... 111

6.6.3 Conclusion ........................................................................ 116

Part II – Domain-Specific Modeling for a Dependable Component-based Software System

7. A Model-Driven Approach to Implementing Dependable Component-Based Mobile Services

7.1 Overview ............................................................................. 117

7.2 Component With Dependability Extensions .................................. 119

7.3 Elements of Component Dependability Extension ............................ 123

7.4 Generating Dependable Applications ......................................... 126

7.5 Building And Deploying Dependable Message-based Mobile Services .......................................................... 128

7.6 Summary ............................................................................. 130

8. A Model-Driven Approach to Implementing Component-Based Real-Time Mobile Services

8.1 Overview ............................................................................. 132
8.2 Modeling Time Constraints ........................................... 136

8.3 Distributed Mobile Services Architecture And Runtime Monitoring of Mobile Services ......................................................... 138

8.4 Building And Deploying Mobile-based Group Communication Services ................................................................. 141

8.5 Summary ........................................................................ 145

9. Conclusion ..................................................................... 146

References ........................................................................ 148

List of Figures
4.1 Eclipse Platform for Tools Integration .................................. 24
4.2 Eclipse Architecture ....................................................... 25
4.3 GMF Work Flow ............................................................ 31
5.1 Model-based Component Composition .................................... 33
5.2 Modeling Component in Meta-model Element ....................... 38
5.3 Modeling Component One-way Reference ............................. 39
5.4 Modeling Component Two-way Reference ............................ 40
5.5 Component Aggregation .................................................. 42
5.6 Component Inheritance .................................................. 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.7 Component Meta-model</td>
<td>45</td>
</tr>
<tr>
<td>5.8 Component Model Instance</td>
<td>46</td>
</tr>
<tr>
<td>5.9 Core Class Hierarchy of GMF Notation Model</td>
<td>53</td>
</tr>
<tr>
<td>6.1 Mobile Service Architecture for Message-based Services</td>
<td>68</td>
</tr>
<tr>
<td>6.2 Message Dispatcher</td>
<td>69</td>
</tr>
<tr>
<td>6.3 Message-based Service Scenario</td>
<td>70</td>
</tr>
<tr>
<td>6.4 Mobile Service Creation Framework</td>
<td>74</td>
</tr>
<tr>
<td>6.5 MSCF Architecture</td>
<td>76</td>
</tr>
<tr>
<td>6.6 MSCF Component Meta-model</td>
<td>79</td>
</tr>
<tr>
<td>6.7 J2ME Web Service Architecture</td>
<td>81</td>
</tr>
<tr>
<td>6.8 Web Service Architecture on MSCF</td>
<td>82</td>
</tr>
<tr>
<td>6.9 Graph Model of MSCF</td>
<td>105</td>
</tr>
<tr>
<td>6.10 Tool’s Definition Model of MSCF</td>
<td>106</td>
</tr>
<tr>
<td>6.11 Mapping Model of MSCF</td>
<td>107</td>
</tr>
<tr>
<td>6.12 Service Model Instance</td>
<td>108</td>
</tr>
<tr>
<td>6.13 A Sample Voting Application</td>
<td>111</td>
</tr>
<tr>
<td>6.14 Voting Sequence Diagram</td>
<td>112</td>
</tr>
<tr>
<td>6.15 Service Model Instance</td>
<td>114</td>
</tr>
<tr>
<td>6.16 Voting Result on a Mobile Phone</td>
<td>115</td>
</tr>
<tr>
<td>7.1 Modeling Component Dependability</td>
<td>120</td>
</tr>
<tr>
<td>7.2 Portion of the Extended MSCF Domain Model With Dependability Specification</td>
<td>122</td>
</tr>
<tr>
<td>7.3 Generated Failure Handler for Components</td>
<td>127</td>
</tr>
</tbody>
</table>
7.3 A Dependable Message-Based Mobile Application Model .........................129
8.1 Real-Time Constraints for Mobile Services...........................................135
8.2 Distributed Mobile Service Architecture..............................................137
8.3 Runtime Support for Real-time Services..............................................139
8.4 A Mobile-based Group Communication Service.....................................142

List of Tables
5.1 Four-Layer Model Architecture..........................................................36
6.1 Attributes of MessagingBasedService..................................................84
6.2 Values of the Attribute ServiceType of the Type MsgServiceType...........85
6.3 Content of MessagingBasedService......................................................86
6.4 Content of ServiceServlet...................................................................87
6.5 References of ServiceServlet.............................................................88
6.6 Attributes of SMSContent..................................................................88
6.7 Attributes of MMSContent..................................................................89
6.8 Attributes of MsgServiceContext.......................................................90
6.9 Attributes of VoteConfig.................................................................91
6.10 Values of the InputFormat Attribute of Type MsgInFormat..................92
6.11 Attributes of WSConfig....................................................................92
6.12 Attributes of DeploymentDescriptor................................................93
6.13 Interfaces of MessageProcessor.......................................................94
6.14 References of MessageProcessor......................................................95
6.15 Attributes of ReminderProcessor.....................................................95
ACKNOWLEDGEMENTS

The writing of this dissertation during the past four years has been one of the most significant academic challenges I have ever had to face. Without the support and patience of the following people, this study would not have been completed.

I would like to gratefully and sincerely thank my esteemed advisor, Dr. Wayne Zage, for his guidance, understanding, and patience, during my graduate studies at Ball State University. His mentorship was paramount in providing a well rounded experience which influenced my life long career goals. He encouraged me to not only grow as a researcher but also as a practitioner.

The members of my dissertation committee, Dr. Jay Bagga, Dr. Mike McGrew, and Dr. Rebecca Pierce, have generously given their time and expertise to better my work. I thank them for their contribution and support.

I must acknowledge as well my friends and colleagues from Motorola software lab for assisting, advising, and supporting my research and writing efforts. Especially, I need to express my gratitude and deep appreciation to Michael Jiang and David Dorenbos who generously shared their insights, knowledge, and wisdoms that supported and expanded my own work.

Finally, the acknowledgements would not be complete without my heartfelt thanks to Dolores Zage who assisted and encouraged me in various ways over the last few years.
Abstract

This study has presented a component-based domain modeling approach that provides an environment for simplifying and accelerating software development and analysis, and improves software reusability, maintainability, and productivity. With high-level design abstraction, constraints of application domains, and the guidance of domain rules, the proposed component-based framework offers an effective solution to modeling and automating the development and deployment of software application. Meta-modeling will be used in this study to define the domain notations, rules, and constraints for component composition within a specific domain context. A domain-specific graphical design environment will also be proposed to simplify and accelerate the software development by simply dragging and dropping pre-built components with minimal programming effort. The modeling of components can be further extended with the specification of their dependability and real-time constraints.
Chapter 1. Introduction

1.1 Motivation

The benefits of using components to build software system are widely acknowledged by industrial organizations. In recent years, the software world has developed the technology and initiated the standards that bring component-based software engineering from the dream stage to practicality [1]. Building new solutions by combining pre-built components improves software quality and enables rapid software development, which leads to shorter time to market. In the mean time, component software encourages the move from the current huge monolithic systems to modular structures that offer the benefits of enhanced reusability, scalability, and maintainability. Once a system is modularized, there is much less need for major release changes and overhaul of the entire system.

Software component technology is one of the most sought-after, and at the same time least-understood, topics in the software field [4]. Although Component-based Software Engineering helps overcome inadequacies in traditional development, it also poses risks to each of its stakeholders [3]. Stakeholders confront challenges in constructing components solutions that address evolving enterprise requirements. These challenges are: encompassing the domain knowledge and its related business process,
determining the granularity of the components, and finding the right components for assembly. Hence, sound strategies are needed for each stage of the component development to ensure the successful use of the component technology.

To address these needs, the research, and number of articles on these topics has grown dramatically. Research results have been reported covering the aspects of component creation, modeling, architecture, repository, composition environment, runtime environment, and so on. As claimed in [3], for component technologies to go much further, domain-specific standards are an absolute requirement. With good understanding of the working domain and the help from domain experts, developers can identify and create components to address a particular task or sequence of tasks shared across the domain. The creation of software applications then becomes the process of assembling components within the domain. Thus the first focus of this dissertation is a Component-based Domain Modeling approach that addresses the need for a simplified software development process with high-quality products.

The second focus of this dissertation is the component composition process for large software systems in a particular domain. According to [5], 50% of bugs are detected after component integration, not during component development. Most future defects in the system being built will arise from mismatches and inadequacies of the composed components. We should also notice that errors in software designs are difficult to find and expensive to correct if propagated to the implementation phase [6]. One of the great challenges for successful component composition is the verification of the correctness of component composition. In traditional software development processes, only syntactical
checks or simple semantic checks, such as for data type, are adopted. These simple checks are often not enough to discover defects caused by structural, functional, or non-functional inadequacies of component composition. The goal of this study is to use meta-models to contain all the syntactic, semantic, and presentation information regarding the domain. The rules defined by the models will be used to govern the construction of component-based software. A domain-specific graphical design environment is also proposed to further simplify and accelerate the software development by simply dragging and dropping pre-built components with minimal programming effort. In this study, mobility service creation is selected as an example domain to elaborate on the development and use of the proposed framework.

To demonstrate the flexibility and extendibility of the proposed framework, I extend the framework with integrated specification of dependability, and real-time constraints. The automatic generation of the protection mechanisms and the automatic generation of the real-time process can minimize component failures, and enforce the real-time assurance of the application. Compared to the previous code-centric approach to developing mobile applications and services, substantial improvement on productivity and dependability is achieved with the integrated framework and its development environment.

1.2 Contributions

This dissertation contributes to the fields of component-based software development and domain-specific modeling in the following three areas.
• Using high-level design abstraction, constraints of application domains, and the guidance of domain rules defined by domain models, the proposed component-based framework will enable rapid creation of software products and support a correct-by-construction component composition process.

Advances in architecture description languages and meta-modeling environments have enabled software developers to hide the complexities of lower level implementation details by defining structural abstractions of components, interfaces, connectors, and system assemblies that can be visualized and analyzed [7]. In this dissertation, I leverage the concept of model-integrated computing [9], a model-oriented technology, for component-based software development. I use meta-models to define the component composition rules and constraints within a specified domain, and use those meta-models to automatically compose a domain-specific design environment. Such a framework can be used as a foundation for providing higher-levels of abstraction in software development that can be leveraged throughout the entire software development process [8]. As opposed to the cumbersome and error-prone textual or XML-based approaches, this approach features a simple graphical notation. Users would benefit greatly from the effective use of visual techniques in the development process. In addition, using the modeling approach to specify the component composition, developers can then selectively add focused semantic information to enable specific kinds of analysis and synthesis. Domain notation and constraints are enforced by the defined domain model, which provides stronger
correctness checks, as opposed to the traditional simple syntactical checks or type checks.

- A visual language, instead of XML syntax, is used for component composition to enable rapid development of applications and services.

By leveraging Eclipse Graphical Modeling Framework (GMF) [10], I present a graphical design environment for rapid application and service creation by dragging and dropping pre-built components from tool’s palettes to graphical editors. This graphical design environment manages diagram persistence, allowing clients focus only on their business logic. With the extensible nature of GMF, the graphical editors are fully open and can be extended by third-parties.

- A highly flexible and extensible architecture for component-based development and analysis enables domain-specific quality and performance specifications.

With the separation of the domain model layer, diagram definition layer, and model mapping layer, this approach enables the isolation of the graphical interface from the syntactic and semantic concerns. Quality and real-time specifications can be integrated with the base domain model to ensure the performance and time constraints of the component-based services. The integrated framework allows the specification of component and service dependability and real-time constraints at the model level independent of languages and platforms. It improves both software quality and productivity.
Chapter 2. Related Work

2.1 Component-based Software Development

Component-based software development (CBSD) focuses on building large software systems by integrating previously existing software components. By enhancing the flexibility and maintainability of systems, this approach can potentially be used to reduce software development costs, assemble systems rapidly, and reduce the spiraling maintenance burden associated with the support and upgrade of large systems [23]. At the foundation of this approach is the assumption that certain parts of large software systems reappear with sufficient regularity that common parts should be written once, rather than many times, and that common systems should be assembled through reuse rather than rewritten over and over. CBSD embodies the "buy before build" and “reuse before buy” philosophy from Fred Brooks [24].

There are various definitions of a software component. The one used herein is defined by Clemens Szyperski’s [3]. *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.* A component can be deployed independently and is subject to composition by third parties. The internal structure of the component is not available to the public.
For software components, the need for standards has been seen for more than a decade. One approach is to build working markets first, followed by the formulation and publication of standards [3]. Microsoft (OLE, ActiveX, COM+, .NET) is the most prominent player following this approach. Another player is Sun Microsystems (Java, JavaBeans, EJB, J2EE). A different approach is to build standards first and then build the markets, such as CORBA [3]. In this study, the component models I will work with are Java-based platforms, such as Java, EJB, and J2EE.

Standards cannot address all the problems of CBSD. In particular, the vast number of domain and application-specific aspects threatens the standardization process [3]. In pursuing a mass-market strategy, developers must identify business areas and domains that generate the yields that would justify component development [4]. Therefore, to improve the yield for a domain, an environment to facilitate the creation of domain-specific software products from the existing components is necessary.

2.2 Domain-Specific Modeling

The way software is developed today is rather primitive and is not as effective as it could be [10]. Programmers frequently do trivial coding to build up a skeleton; this is tedious and error prone. Software developed from a manually-built skeleton also becomes a maintenance burden, since a change in one place (e.g. addition of a new property/type) will ripple into several other classes, interfaces or configuration files [10]. One of the popular solutions is to use Domain-Specific Modeling (DSM) that distills domain knowledge into a meta-modeling language [11]. It raises the level of abstraction beyond
programming by specifying the solution directly using domain concepts. For example, DSM for mobile application would have concepts like “Receiving SMS/MMS”, “Sending SMS/MMS”, and processors for processing the received messages. Symbols in a domain-specific model map to things in the domain - the world in which the application is to run. This is a level of abstraction higher than UML, and makes each symbol worth several lines of code. The application can be automatically generated from these high-level specifications with domain-specific code generators, aided where necessary by existing component code. Clearly, when the generated code is based on the expert’s knowledge of a particular domain, quality is going to increase. In addition, since the application is specified by models at a higher level of abstraction, different generators can generate different application versions for different languages. The key is that only one center or team is needed to create and maintain the generators, all of the modelers receive the benefits for all the models.

There is one general attempt at achieving the DSM - OMG's MDA initiative [12]. There are tools from several vendors that support this effort, such as CodaGen, Rational Suite, and I-Logix Rhapsody. Using MDA, developers define platform-independent models (PIMs) that are made domain-specific using a UML profile. This is not a new idea, but it can be difficult to make it work without introducing too much complexity. In addition, the direct DSM approach leads more naturally to modularity of expression than the UML profiling approach.

Another approach is to create real domain-specific languages. There are a few vendors with offerings in this field, such as MetaCase [14], and an open source product
developed by the Institute for Software Integrated System (ISIS) at Vanderbilt University called Generic Modeling Environment (GME) [13]. GME has been applied to modeling and synthesizing several real-world applications for both the U.S. government and industry organizations. However, as Steven Kelly, one of the founders of DSM Forum, has said, "GME is more for academic investigation than a polished commercial product".

The modeling environment I have adopted in this study is based on the Eclipse Modeling Project (EMP) [15], especially its subprojects Eclipse Modeling Framework (EMF) [16], Graphical Modeling Framework (GMF) [17], and Object Constraint Language (OCL) [18]. In section 4.4, I will describe in detail how these frameworks can contribute to the component-based software development process.

2.3 Domain-specific Modeling and Software Component

Both DSM and CBSD approaches have their advantages as well as disadvantages. Combining the two approaches can overcome some of the disadvantages of each and enable the faster creation of software products. To achieve this combination, I use metamodels to specify components, component composition rules and constraints, and domain notations of a component-based framework. With this approach, components can be visually described by the model elements, and communications rules and constraints between components are enforced by the connections between model elements. The meta-models are used to generate the target domain-specific component composition environment. The generated environment is then used to build component composition instances, which can automatically generate applications or synthesize input to other
development or analysis tools. GME [13], MetaEdit+ [35], and the recent player Microsoft DSL tools [36] provide nice environments for creating a domain-specific language based on DSM technology, however, none of these frameworks focus on CBSD.

By using the OCL to attach constraints to meta-models, the proposed approach can capture more syntactic and semantic information of a specified domain, so that stronger correctness checks will be enforced during the construction of the component-based products. In the traditional component-based software development process, only syntax checks and simple semantic checks, such as a data type check, are provided [20][22]. In [21] the authors proposed composition rules allowing for powerful static correctness checks at composition time. However, the proposed rules don’t include the richer semantic information obtained through OCL. Moreover, the textual rules are more cumbersome and error-prone comparing to the modeling approach. I believe that the proposed approach provides a more powerful means to support a component-based software development process.

The implementation of this approach is based on Eclipse [19], an open platform for tool integration through extension points defined by its plugins, written in Java, developed by an open community. The extensive plugin support allows easy development of custom application plugins supporting specific application functionalities. The EMF is one of the most successful Eclipse projects that allows users to generate executable code from meta-models, called.ecore models in EMF and defined as by simplified UML class diagrams. In this dissertation, a customized EMF framework will be used to build domain meta-models. To further facilitate and accelerate software development, the Graphical
Modeling Framework (GMF) [17] is adopted to create a user-friendly graphical component-based design environment, in which a user can compose the domain components by means of drag and drop. Based on GMF, I define diagram definition models to define graphical elements to be displayed in the graphical design environment. A diagram mapping model is used to define mappings between domain model elements and graphical elements, so that the final composition instances constructed in the graphical editor are able to capture the notations, rules and constraints defined in the domain meta-model.

2.4 Dependable Component-Based Software Systems

A precondition for successful use of CBD is the utilization of theories, methods and technologies to predict and evaluate dependability attributes [62]. The requirements for dependability are increasingly important for general-purpose systems. Dependability is an increased concern in software development in general, as well as in CBD. The original definition of dependability [64] for a computing system gathers the following attributes or non-functional requirements: Availability: availability refers to the ability of the user community to access the system, whether to submit new work, update or alter existing work, or collect the results of previous work. If a user cannot access the system, it is said to be unavailable [66].

- **Reliability**: Software reliability is an important facet of software quality. It is defined as "the probability of failure-free operation of a computer program in a specified
environment for a specified time" [67]. One of reliability's distinguishing characteristics is that it is objective, measurable, and can be estimated, whereas much of software quality consists of subjective criteria [68]. Without software reliability, companies lose credibility and risk financial loss due to software errors and bugs. Implementing a code quality mindset early in the application development lifecycle will improve software integrity and boost customer satisfaction.

- **Maintainability**: In software engineering, software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

- **Safety**: assures that a life-critical system behaves as needed even when pieces fail. Software safety is a fast growing field since complex, physical systems are increasingly being put under control of software. The whole concept of system safety and software safety, as a subset of systems engineering, is to influence safety-critical systems designs by conducting several types of hazard analyses to identify risks and to specify design safety features and procedures to strategically mitigate risk to acceptable levels before the system is certified [69].

- **Real-Time**: A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed. The classical conception is that in a hard or immediate real-time system, the completion of an operation after its deadline is considered useless - ultimately, this may lead to a critical failure of the complete system. A soft real-time system on
the other hand will tolerate such lateness, and may respond with decreased service quality (e.g., dropping frames while displaying a video) [70].

These qualities are the shared concern of many sub-disciplines in software engineering such as computer security, reliability and safety engineering. Most of these properties are of interest for software-intensive systems. When applying CBD, questions such as transformation of components to execution entities (tasks or threads), component specification of real-time properties, and composition of real-time properties become important issues to achieve system correctness.

We are witnessing an increasing effort in research and in the industry to combine CBD with modeling and analysis methods and tools for dependable systems. Actually, many modeling and analysis theories and tools exist, but are not integrated in CBD. For this reason, the most important initiative is bringing these worlds together, making researchers and practitioners aware of both sides. By integrating the model-driven engineering approach to model both dependability and real-time constraints, the proposed framework effectively improves software productivity and quality. Software components are represented as formal model constructs to enable the specification of their composition, interactions, and dependability. A model-based development environment is developed for the creation of domain-specific models and application code is generated from the domain models. The generated code implements the functions of domain applications, real-time constraints, and the handling of components failures to improve the overall system dependability.
Part I

Domain-Specific Modeling for Component Composition
Chapter 3. Component Composition

3.1 CBSD Overview

Component-based software development (CBSD) is concerned with the assembly of pre-existing software components into larger pieces of software. Underlying this process is the notion that software components are written in such a way that they provide functions common to many different systems. Borrowing ideas from hardware components, the goal of CBSD is to allow parts (components) of a software system to be replaced by newer, functionally equivalent, components. The promise of CBSD is a reduction in development costs: component systems are flexible and easy to maintain due to the intended plug-and-play nature of components.

CBSD consists of the following major activities:

- Requirement analysis
- Component model selection and creation

A component model defines a set of standards for component implementation, naming, composition, evolution, and deployment. A component model also defines standards for an associated component model implementation, the dedicated set of
executable software entities required to support the execution of components that conform to the model [1].

There are numerous component models currently available. The main component models are OMG CORBA Component Model (CCM), Microsoft (D)COM/COM+ family, and SUN Microsystems’ JavaBeans and Enterprise JavaBeans. The selection of a right component model can bring the compliant components into full play. It is not suggested to have too many standards, but it is necessary to have specialized standards for describing domain-specific features required for certain applications.

- Component selection

Component selection is the process of determining suitability of a component for use within the intended domain. The issues of trust and certification arise during this activity. The process of certification is two-fold [41]:

- to establish facts about a component and to determine that the properties a component possesses is also conformant with its published specification; and

- to establish trust in the validity of these facts, perhaps by having a trusted third-party organization attest the truth of this conformance and provide a certificate of verification.

The motivation for component certification is creation of a causal link between component’s certified properties and the properties of the end system. If enough is
known about the certified components selected for assembly, it may be possible to predict the properties of the final assembled system. Accuracy of the prediction is founded on the degree of trust in a component and also how well the glue that connects the components is understood. The prediction can be difficult due to the lack of information about a component’s capabilities and lack of trust in this information.

- Component composition

Component composition is the combination of two or more software components yielding a new component behavior at a different level of abstraction. The characteristics of the new component behavior are determined by the components being combined and by the way they are combined [1].

- Component deployment

Components are units of deployment. A component model must be able to describe how the components are packaged, so that they can be deployed independently. It is required that deployment of components (including installation, configuration, and de-installation) does not have side-effects that affect the behavior of the components. A deployment description provides information about the contents of a package (or of a number of related packages) and other information that is necessary for the deployment process. This description is analyzed by the target component infrastructure and used for installing and configuring a component properly [1].
In this study, I will focus on the composition stage of the component-based software development. Component composition enables prefabricated “things” to be reused by rearranging them in ever-new composites [3]. According to [46], 50% of bugs are detected after component integration, not during component development. Therefore, a robust component composition infrastructure is crucial for software development.

### 3.2 Approaches of Component Composition

Various approaches to component composition at different levels of abstraction have been identified [42]. The most used approaches for component composition are described as following [1]:

- **all-purpose programming languages**

  This approach uses programming languages, such as C++ and Java, to support component composition. Composition rules and constraints are defined at low level programming language. Rearrange of the components will lead to the change at code level, which makes the system less flexible and reusable.

- **scripting or glue languages**

  Glue languages, such as PHP, PERL, JavaScript, and TCL, are programming languages used for connecting software components together. They support component composition at a higher level of abstraction than all-purpose programming languages.
• visual programming or composition tools

A Visual programming language (VPL) [43] is a programming language that lets users specify programs by manipulating program elements graphically rather than by specifying them textually. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols. Most VPLs are based on the idea of "boxes and arrows," that is, boxes or circles or bubbles, treated as screen objects, connected by arrows, lines or arcs.

VPLs may be further classified, according to the type and extent of visual expression used, into icon-based languages, form-based languages, and diagram languages. Visual programming environments provide graphical or iconic elements which can be manipulated by users in an interactive way according to some specific spatial grammar for program construction.

Composition through visual programming raises further the level of abstraction. However, there are drawbacks to visual approaches such as lack of density, structure of graphical representations and the extra effort needed for graphic editing and layout [44].

• component infrastructure

Maximum reuse is achieved with a component infrastructure designed for a specific domain where the interaction among components is predefined. Composition with a component infrastructure is a matter of inserting and substituting components conforming to the action standards defined by the infrastructure. A good component
infrastructure enables not only the reuse of individual components but of an entire design. A concrete example of the power of such infrastructure is SOA. However, such component infrastructure typically requires many software products to implement. It becomes a big challenge to understand the inter-system dependencies.

Different from the traditional component composition approaches, this study leverages meta-modeling to visually define the composition rules and constraints of the compliant components in a particular domain. Domain-specific design environments capture specifications and automatically generate and configure the target applications in particular engineering fields. Applications developers need only focus on the business logic of the target application with little understanding of how the components are connected by the backend model-based structure. Details of the design are elaborated in Chapter 5.
Chapter 4. Domain-Specific Modeling

4.1 What is domain-specific modeling?

Domain-Specific Modeling (DSM) [46] is a software engineering methodology for designing and developing software systems. It involves systematic use of a graphical Domain-Specific programming Language (DSL) to represent the various facets of a system. DSM languages tend to support higher-level abstractions than general-purpose modeling languages, such as UML, XML, and LISP, so they require less effort and fewer low-level details to specify a given system.

DSM often also includes the idea of code generation, that is, the creation of executable source code directly from the DSM models. Being free from manual creation and maintenance of source code means DSM can significantly improve developer productivity. The reliability of automatic generation compared to manual coding will also reduce the number of defects in the resulting programs, thus improving quality.

DSM differs from earlier code generation attempts in the CASE tools of the 1980s or UML tools of the 1990s. In both of these, the code generators and modeling languages were built by tool vendors. While it is possible for a tool vendor to create a DSM language and generators, it is more normal for DSM to occur within one organization. One or a few expert developers create the modeling language and generators, and the rest
of the developers use them. Having the modeling language and generator built by the organization that will use them allows a tight fit with their exact domain and needs. It also reduces the time needed for developers to learn the modeling language, since it can use familiar terms and concepts. Finally, since only one organization's requirements need be taken into account, it is easier for the modeling language to evolve in response to changes in the domain.

Most existing DSM takes place with DSM environments, either commercial such as MetaEdit+, open source such as GEMS, or academic such as GME. The increasing popularity of DSM has led to DSM frameworks being added to existing IDEs, for instance in the Eclipse Modeling Project (EMP) with EMF and GMF, or in Microsoft's DSL Tools for Software Factories.

4.2 Why domain-specific modeling?

CBSD means different things in different domains. The specific nature of different types of applications places different requirements on CBSD. For component technologies to go much further, domain-specific standards were an absolute requirement [3]. A developer encounters certain risks and challenges in composing components. In pursuing a mass-market strategy, developers must identify business areas or domains that would generate sufficient component reuse to justify the development of component framework.

According to [47], DSM shows 5-10 times greater productivity than current software development practices. Domain-Specific Modeling raises the level of
abstraction and hides general purpose programming languages, like C++/C and Java. With DSM, expert developers define a domain-specific language containing the chosen domain's concepts and rules, and specify the mapping from that to code in a domain-specific code generator. The other developers then make models with the modeling language, guided by the rules, and code is automatically generated. As an expert has specified the code generators, they produce products with better quality than the ones produced by hand.

4.3 Domain-specific modeling vs. UML

Increasing complexity of software is leading many companies to add a modeling phase to the development processes [48]. Currently the best-known means for making designs is Unified Modeling Language (UML), which aims to provide a visual representation of the design. While designing before beginning the implementing makes sense, companies often want more from the models than just a throwaway specification or documentation that often ends up not reflecting what is actually built. Automatically generating production code from the UML designs would provide a solution, but this is where the UML generally falls short: In practice, it's possible to generate only very little usable code from UML models. Developers still need to add or edit the majority of the code by hand.

Unlike UML, DSM provides modeling constructs that mimic the real-world objects more closely. Most importantly, it provides complete code-generation capabilities. Together, these lead to a much more productive way for developers to build applications.
4.4 Eclipse-Based DSM Framework

With the increasing popularity of DSM, it’s framework has being added to existing IDEs, for instance in the Eclipse Modeling Project (EMP) with EMF and GMF, or in Microsoft's DSL Tools for Software Factories. In this study I extend the open source Eclipse platform, especially the Eclipse Modeling Framework (EMF) [16], and Graphical Modeling Framework (GMF) [17] subprojects to build the domain-specific modeling CBSD framework. To better understand the proposed framework and how it leverages the extensible and flexible Eclipse platform, I will give a brief introduction of Eclipse and its architecture in section 4.4.1 and 4.4.2. Eclipse Modeling Framework and Eclipse Graphical Modeling Framework will be introduced in section 4.4.3 and 4.4.4.

4.4.1 Eclipse Overview

Eclipse is an open source platform for tool integration built by an open community of tool providers, with key contributions from IBM, Borland, and others. It has become one of the most flexible, powerful, and integrated Java development environments. Further, many highly visible commercial products are built on Eclipse. Significant efforts have been put, and continue to be put, into the Eclipse consortium to evolve it into a platform that covers the full system development life cycle: modeling, design, development, and testing. Examples include Eclipse’s UML2 project, EMF modeling framework, and Automated Software Quality (ASQ) tool for testing. Eclipse’s WSVT (Web Service Validation Tools) project aims to provide a set of tools for the development, deployment, testing and debugging of web-service applications. The
Eclipse Web Tools Platform project provides a set of common frameworks and services to allow tool developers to easily build highly integrated tools that work with web application servers that support the J2EE specification. The Generative Model Transformer project is targeted for model driven software development. Together, the Eclipse platform and its integrated tools form a complete environment for software design, development, and testing.

The Eclipse platform defines the set of frameworks and common services that collectively make up "integration-ware" required to support a comprehensive tool integration platform. These services and frameworks represent the common facilities required by most tool builders. Many plug-in tools are available from the Eclipse Foundation, various open sources, and commercial vendors. Except for the small Eclipse runtime kernel, all the other components are plug-in tools integrated into the Eclipse platform through pre-defined extension points.
As a universal tools development platform, Eclipse can be used to develop and maintain deliverables throughout the development life cycle, as shown in Figure 4.1. By using the corresponding plug-ins, developers don’t have to switch between tools to work on various phases of the project and to manage the interactions of these tools and deliverables produced by these tools.

4.4.2 Eclipse Architecture

![Eclipse Architecture Diagram]

Figure 4.2: Eclipse Architecture

Figure 4.2 shows the basic structure of the Eclipse platform [1]. Fundamentally, Eclipse is a framework for plug-ins. Besides its runtime kernel, the Eclipse Platform consists of the workbench, workspace, help, and team support. Other tools plug into this basic framework to create a usable application. Except for the runtime kernel, everything
in Eclipse is implemented as plug-ins. Plug-ins add functionality to the platform by contributing to pre-defined extension points. The workbench user interface is implemented by a group of such plug-ins. When we start up the workbench, we are not starting up a single Java program. We are activating a platform runtime, which can dynamically discover registered plug-ins and start them as needed.

When we want to provide code that extends the platform, we accomplish this by defining system extensions in our plug-ins. The platform has a well-defined set of extension points where you can hook into the platform and contribute to system behavior. From the platform's perspective, a custom plug-in is not different from basic plug-ins like the resource management system or the workbench itself. The plug-in architecture of the Eclipse platform provides a flexible, open, and scalable solution to tool integration. With the open Eclipse platform, a component can be customized or extended to suit the needs of the development environment. For example, if a perspective is inadequate, you can create your own perspective or plug in an appropriate one available from the Eclipse open source community or commercial providers. Once we have created a plug-in in our workspace, we can easily deploy the plug-in using the export wizard. Eclipse also conveniently supports software updates. New plug-ins and updates to existing ones can be delivered by creating an “update site project” through the Eclipse Plug-in Development Environment (PDE). The following is a brief description of each component.

• Workspace – Responsible for managing the user’s resources, which are organized into one or more projects at the top level.

• Workbench – Graphical user interface.

• Team Support – Version control (or configuration management) system

• Help – Extensible documentation system.

4.4.3 Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. When developing a model, we generally think about things like Class Diagrams, Collaboration Diagrams, State Diagrams, and so on. Unified Modeling Language (UML) defines a standard notation for these kinds of diagrams. Using a combination of UML diagrams, a complete model of an application can be specified. This model may be used purely for documentation or, given appropriate tools, it can be used as the input from which to generate part or all of an application. An EMF model requires a small subset of the diagrams that you can model in UML, specifically, simple definitions of the classes and their attributes and relations for which a full-scale graphical modeling tool is unnecessary.

While EMF uses XMI (XML Metadata Interchange) as its canonical form of a model definition [1], there are several ways of getting models into that form:
• Create the XMI document directly, using an XML or text editor.

• Export the XMI document from a modeling tool such as Rational Rose.

• Annotate Java interfaces with model properties.

• Use XML Schema to describe the form of a serialization of the model.

Once we specify an EMF model, the EMF generator can create a corresponding set of Java implementation classes. We can edit these generated classes to add methods and instance variables and still regenerate from the model as needed. Our additions will be preserved during the regeneration. If the code we added depends on something that we changed in the model, we will still need to update the code to reflect those changes; otherwise, our code is completely unaffected by model changes and regeneration.

In addition to simply increasing productivity, building applications using EMF provides several other benefits like model change notification, persistence support including default XMI and schema-based XML serialization, a framework for model validation, and a very efficient reflective API for manipulating EMF objects generically. For instance, from an EMF class diagram, EMF generates a set of Java classes for manipulating the model and a basic, tree-based editor for model instances. The generated classes provide basic support for creating/deleting model elements and persistency operations like loading and saving.

EMF consists of two fundamental frameworks:
• EMF core framework -- provides basic generation and runtime support to create Java implementation classes for a model.

• EMF.edit -- includes generic reusable classes for building editors for EMF models. It provides:
  
  o Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.
  
  o A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.
  
  o A code generator capable of generating everything needed to build a complete editor plug-in for your EMF model. It produces a properly structured editor that conforms to the recommended style for Eclipse EMF model editors. You can then customize the code however you like, without losing your connection to the model.

4.4.4 Eclipse Graphical Modeling Framework

GMF is a framework that is a bridging technology between the GEF platform and the EMF modeling platform. GEF provides a platform for building graphical editors and EMF is a modeling technology for managing and persisting data at a higher level of abstraction (see 4.4.3). The bridge provided by GMF links these technologies seamlessly so that the graphical editor displays data stored and managed by EMF.
There are usually two aspects to the data stored in a graphical editor. The first is the diagramming data that represents the shapes and connections displayed in the editor (otherwise known as notation), and second is the semantic data that the user is editing. From the graphical editor user’s point of view, the two are synonymous – the notation view is simply a window of the semantic data. However, there is an important distinction between the notations and semantics that makes it necessary to separate them. The semantic data can be displayed in different contexts or editors and as such shouldn’t store information about how it can be displayed in an editor. Also, the semantic data can potentially be displayed multiple times in the same diagram which means the notation information needs to be stored multiple times. In order to facilitate this, it is necessary that the notation information be stored in a different model (defined by a notation meta-model) that references the semantic model.

As displayed in Figure 4.3, GMF has a set of models to be created to generate a graphical editor.

- graphical definition, which defines the visual aspects of our generated editor. This model:
  - Creates the figures to be displayed on the diagram.
  - Creates the nodes seen on the diagram
  - Creates connections on the diagram.
  - Assures each node matches up to the figure created in the previous step.
tooling definition, which comprises elements related to editor palettes, menus, etc.

Finally, mapping definition, which defines the mapping between the business logic (the EMF ecore model) and visual models (graphical and tooling definition).

The above design allows multiple clients of GMF to have diagram interoperability through a compatible and consistent notation format. Additionally, since the notation is generic, it allows management of the notation data to be encapsulated from clients and lets them focus their development efforts on the business logic. In the meantime, the separation of notation and semantics allows clients to define multiple notations for the same semantic element.

Figure 4.3: GMF Work Flow

The above design allows multiple clients of GMF to have diagram interoperability through a compatible and consistent notation format. Additionally, since the notation is generic, it allows management of the notation data to be encapsulated from clients and lets them focus their development efforts on the business logic. In the meantime, the separation of notation and semantics allows clients to define multiple notations for the same semantic element.
It's important to note that generating a graphical editor is only a fraction of GMF's capabilities. There is much work that can be done to take advantage of the advanced features of the framework. For example, GMF has support for validation by taking advantage of the Eclipse Modeling Framework Technology (EMFT). If there are violations, GMF will put error markers in files that don't pass validation similar to what a Java compiler does for Java files that don't compile.
Chapter 5. Model-based Component Composition

Environment for a Specific Domain

5.1 Overview

To get to a situation of DSM with full automatic code generation, the proposed model-driven development framework consists of three main layers: component library, domain meta-model, and model instance editor. As what has been displayed in Figure 5.1,

![Figure 5.1: Model-based Component Composition](image)

the lowest layer of the framework is composed of service component library. The components in the library are called by the model elements defined in the second layer.
Each domain element itself represents a particular domain component whose implementation can be generated automatically by the domain model framework. The components presented in the domain model can contain or refer to any model components generated by the framework or the components preloaded in the component library. The relationships of components are specified as connections in the domain model. Java code will be generated by the framework to provide API access to each component defined in the model. The top layer of this framework provides developers an interface to build applications by composing the model components through a graphical editor. Composition rules and constraints are enforced by the definition of the underlying domain meta-model.

### 5.2 Domain meta-model

To make the component composition framework DSM-capable, I propose to use domain specific meta-models to define the composition rules between components. So what is a meta-model? In short, a meta-model is a precise definition of the constructs and rules needed for creating semantic models [72]. For many times, when we model a set of related systems, usually belonging to a given domain, we realize that these models share many constructs. We are then able to generalize across these different models and come up with a model to which the set of related models should conform. This is what we call a “meta-model”, a model of models [71].

Meta-modeling has been around at least since the late 1980s, but it started to get more attention just recently for the following reasons [72]:

• with the advent of the Internet and business integration, data integration is obviously a first-order priority. Meta-models are the foundation for data integration.

• The increasing availability of meta-model-driven technologies and standards.

• The need to raise the abstraction level.

Meta-models are very good at abstracting from lower-level details of integration and interoperability, and at helping with partitioning problems into orthogonal sub-problems of conceptual data, physical data optimization, and control flow. In this respect, meta-models are an ideal helper for complex projects.

But what exactly is the difference between a model and a meta-model? Is a meta-model a model? Is there such a thing as a meta-metamodel? In [73], the author explains the definition of model, meta-model, and meta-metamodel in the context of Model Driven Engineering (MDE) and UML [73].

As explained in Table 5.1, a meta-metamodel defines the rules for defining a meta-class. This level of abstraction supports the creation of many different models from the same set of basic concepts. A meta-model is a model that describes the artifacts of and rules for a model. A model is the layer that describes the artifacts and rules for the problem domain. This is the level at which we draw Class diagrams, Sequence diagrams, and so forth in the context of UML. Classes and objects, associations, attributes, and all other elements of the model layer depend on the meta-model definitions. User object consists of the runtime elements created by executing the model.
<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-metamodel (M3)</td>
<td>The infrastructure for a meta-modeling architecture. Defines the language for specifying meta-models.</td>
</tr>
<tr>
<td>Meta-model (M2)</td>
<td>An instance of a meta-metamodel. Defines the language for specifying a model.</td>
</tr>
<tr>
<td>Model (M1)</td>
<td>An instance of a meta-model. Defines a language to describe an information domain.</td>
</tr>
<tr>
<td>User Object (User Data) (M0)</td>
<td>An instance of a model. Defines the values of a specific domain.</td>
</tr>
</tbody>
</table>

Table 5.1: Four-Layer Model Architecture

It is important to understand that meta-models are always made for a particular purpose. We should never attempt to use a meta-model without understanding the particular goal of the meta-model when it was created. We always need more than one meta-model for all modelers because the constructs and rules used to model one type of applications (i.e. mobile services) are different than the constructs and rules used to model other types of applications (i.e. data structures). In the proposed framework, I introduce a meta-model for the domain of mobile service creation. The model elements are used to define domain components, and model connections represent composition rules and constraints between components. Section 5.2.1 describes how to use meta-model to model properties and interfaces of a component. Section 5.2.2 explains how to
use different types of model connections to represent relationships between components. Modeling aggregation and inheritance of the domain components are discussed in Section 5.2.3.

### 5.2.1 Modeling Component

There are two types of components in the proposed framework: predefined components from the component library and the components defined by the meta-model elements. This section focuses on using model elements to define component interfaces and properties. In the proposed model-based component framework, a corresponding component is generated by the framework based on the definition of each model element. The component is depicted as boxes with three sections in a meta-model diagram. The top one indicates the name of the component, the middle one lists the properties of the component, and the third one lists the interfaces. Figure 5.2 shows the model representation of component `DSMComponent`, which has one String type property `attr`, and one interface, `fun()`, with the return type String. The declared properties and interfaces are the channels for `DSMComponent` communication with other components. Figure 5.2 shows the corresponding component diagram in the context of UML. Property `attr` is interpreted as providing two interfaces in the UML component diagram: getAttr() and setAttr().
The Java implementation of DSMComponent

```java
public interface DSMComponent {
    String getAttr();
    void setAttr(String value);
    String fun();
}
```

5.2.2 Modeling Component Association

One-way reference

Figure 5.3 expands the model in Figure 5.2 with another component, DSMComponent2, which has a single one-way association with DSMComponent1. The reference name used for DSMComponent1 to access DSMComponent2 is “ref”. The
number “1” of the reference implies that only one DSMComponent2 object can be returned in an instance of DSMComponent1.

**Figure 5.3: Modeling Component One-way Reference**

**Java Implementation of Figure 5.3:**

```java
public interface DSMComponent1 {
    String getAttr();
    void setAttr(String value);
    DSMComponent2 getRef();
    boolean isContainment(DSMComponent2); 
    String fun();
}

public interface DSMComponent2 {
    int getAttr1();
    void setAttr1(int value);
    int getAttr2();
    void setAttr2(int value);
    Object fun1();
}
```
Two-way reference

The association in Figure 5.4 is now two-way, as indicated by the lack of an arrowhead on the DSMComponent2 end of the association line. When setting one end of a two-way reference, the other end of the reference needs to be set as well. In figure 5.4, an instance of DSMComponent2 can get a list of DSMComponent1 instances via ref1. In the mean time, an instance of DSMComponent1 can get exactly one DSMComponent2 instance via ref2. The Java implementation of this relationship is very similar to a one-way reference, except that component DSMComponent2 has a method getRef1() to return a list of DSMComponent1 instances.

Figure 5.4: Modeling Component Two-way Reference
Java Implementation of Figure 5.4:

```java
public interface DSMComponent1 {
    String getAttr();
    void setAttr(String value);
    String fun();
    DSMComponent2 getRef2();
    boolean isContainment(DSMComponent2);
}

public interface DSMComponent2 {
    int getAttr1();
    void setAttr1(int value);
    int getAttr2();
    void setAttr2(int value);
    Object fun1();
    List<DSMComponent1> getRef1();
    boolean isContainment(DSMComponent1);
}
```
5.2.3 Modeling Component Aggregation and Inheritance

Inheritance and object composition are two techniques for reusing functionality in object-oriented systems [49]. The first technique, class inheritance, or sub-classing, allows a subclass' implementation to be defined in terms of the parent class' implementation. This type of reuse is often called white-box reuse.

In the second technique, objects are composed to achieve more complex functionality. Because objects are treated only as "black boxes," this type of reuse is often called black-box reuse. This study borrows these two important relationships, aggregation and inheritance, from object-orientate language. Aggregation, representing $1 \rightarrow N$ relationships between components, corresponds to the nesting of such components inside one another. A container component can contain all aggregated components as children. This way, we can combine simple components into more complex ones making this technique a very critical building block of component-based structure. Figure 5.5 shows an aggregation relationship between DSMComponent1 and DSMComponent2. The aggregation reference is represented as a black diamond on the connector.

![Component Aggregation Diagram](image_url)
The Java implementation of this relationship is very close to the implementation of a one-way reference we have discussed. The only difference is that in \textit{DSMComponent1}, the method \texttt{isContained()} always returns true for the aggregation relationship.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{figure5_6}
\caption{Component Inheritance}
\end{figure}
Java Implementation of Figure 5.6

```java
public interface DSMComponent2 extends DSMComponent1{
    int getAttr1();
    void setAttr1(int value);
    int getAttr2();
    void setAttr2(int value);
    Object fun1();
}
```

Inheritance is used for improving the clarity of the model diagrams, as the common attributes and relationships shared among multiple components can be abstracted into one. Thus, we can avoid describing the same relationships and attributes more than once. In the actual document, only the concrete components are instantiated into the corresponding elements, where all the attributes and behaviors of the ancestor components have been moved downwards along the hierarchy and inherited by the leaf components. Figure 5.6 shows an example of component inheritance. When one component (DSMComponent2) extends another component (DSMComponent1), the extended component inherits all of the metadata and behavior from the parent it extends. In the extended component, we can add new features or customize existing features of its parent component the same way as what we can do to objects in OO languages.
5.3 Model Instance

As an example of applying the previous rules, Figure 5.7 shows a meta-model which includes all the relationships described above.

Figure 5.7: Component Meta-model
In Figure 5.7, “Component” inherits all the attributes from “Abstract-Component”, has two extended attributes “attr1” and “attr2”, and one extended interface fun1(). Component has a one-way reference to Ref-Component1 and a two-way reference to Ref-Component2. Component also contains multiple Child-Components, which means that the instance of Child-Component can only exist within a component instance. Ending the life cycle of a Component instance will also end the life cycle of the contained Child-Components.

![Component Model Instance](image-url)
Figure 5.8 shows a sample model instance created based on the component meta-model demonstrated in Figure 5.7 using Eclipse GMF editor. As discussed in Section 4.4.4, the graphical and semantic data are separated but users manage the semantic data through the property page of the graphical diagram editor.

5.4 Code Generation

Code generation is a very important phase in a model-based development framework. In this phase the expert specifies how code is automatically generated from the structures in users’ models.

DSM says that developers can design software more effectively if they use a notation tailored to fit the problem domain at hand. It should allow developers to capture all the needed aspects of the application. Provided this notation and the way it maps to code is defined by a company's expert, the rest of the developers will be able to automatically generate most or full, top-quality code from their designs. The DSM tool should provide the necessary functionality for creating such generation scripts, and should guide the experts by allowing them to reference and use the concepts in the metamodel. With companies like Microsoft, MetaCase and Xactium and frameworks like Eclipse GMF providing support for DSM, the DSM support is available already.

Of all the phases, code generation probably varies the most among domains. In some, it will be possible to produce a large fraction of code with a relatively simple code generation scripting language, such as is already provided in most DSM toolsets. In other domains, it may be necessary to use a more powerful language to operate on data
exported from the modeling tool. The most important goal is that the end-user should be able to use the code generation simply. In either case, the cost of developing the code generation is defrayed over many users.

Implementation of DSM is not an extra investment if we consider the whole cycle from design to working code, rather, it saves development resources. Traditionally, all developers work with the problem-domain concepts and map them to the implementation concepts manually - and among developers, there are big differences. Some do it better, but many not so well. If an expert specifies the code generator, it produces applications with better quality than it could be achieved manually.

### 5.4.1 Model code generation

My study leverages and extends the EMF Framework to generate model code for component meta-models. In the generated model code, each component corresponds to two things in Java: an interface and a corresponding implementation class. For example, the Java interface for \texttt{DSMComponent} looks like:

```java
Public interface DSMComponent ...
```

and the corresponding implementation class looks like:

```java
Public class DSMComponentImpl extends ... implements DSMComponent
```

A factory class is used to create a new component instance as shown in the following:
Each generated interface extends directly or indirectly from the base EMF interface EObject:

```java
Public interface DSMComponent extends EObject{
...
}
```

EOBJECT is the EMF equivalent of Java.lang.0bject, that is, it is the base of all modeled objects. Extending EObject introduces behaviors like: returning object’s containing object or resource, providing APIs for accessing objects reflectively, model change notification, and so on.

- **The reflective EObject API**

  eGet(), eSet(), eIsSet(), and eUnset() in EObject provide APIs to manipulate component instances reflectively. For example, I can set the `attr` attribute of an instance of `DSMComponent` like this:

  ```java
  aDSMComponent.eSet(attrAttribute, “String Value of attr Attribute”);
  ```

  It can be read back like this:

  ```java
  String shipTo = (String) aDSMComponent.eGet(attrAttribute);
  ```

  In addition to the eGet() and eSet() methods, the eIsSet() method can be used to find out if an attribute is set or not, while eUnset() can be used to unset or reset it.

- **Notification**
Since EObject also extends interface Notifier, every generated component class is also a Notifier, that is, it can send notification whenever an attribute or reference is changed. It allows Components to be observed, for example, to update views or other dependent objects.

Notification observers in EMF are called adapters. As a simple observer, an adapter can be attached to any component object by adding to its adapter list like this:

```java
Adapter aCompObserver = ....
aDSMComponent.eAdapters().add(aCompObserver);
```

After doing this, the notifyChanged() method will be called whenever a state change occurs in `aDSMComponent`.

- **Object Persistence**

The ability to persist and communicate with other persistent model objects is one of the most important features for model-based frameworks. Models can be serialized using a default EMF XMI serializer. Beyond the default XMI serializer, EMF allows users to write their own serialization code so that users can save objects in any persistent form they like.

To make objects persistent, we need to put them into a resource, an interface used to represent a physical location. We can get a resource from the factory interface `ResourceSet` via `createResource()` method. All we need to do is to add the root object to a resource returned from a `resourceSet` and then save the resource as an XMI file to
the defined URI location. Sample code for persisting a DSMComponent object is as following:

```java
ResourceSet aResourceSet = new ResourceSetImpl();

URI saveLocation = URI.createFileURI(new File("dsmCompSource.xml").getAbsolutePath());

Resource aResource = aResourceSet.createResource(saveLocation);

aResource.getContents().add(aDSMComponent);

aResource.save(null);
```

Loading a resource is similar to saving component objects. The differences between them are highlighted as bold in the code below:

```java
ResourceSet aResourceSet = new ResourceSetImpl();

URI saveLocation = URI.createFileURI(new File("dsmCompSource.xml").getAbsolutePath());

//load resource from the defined location.

Resource aResource = aResourceSet.getResource(saveLocation, true);

//return root object of the model
```
5.4.2 Editor Code Generation

The component model APIs are independent of the UI that supports reuse of the model code. This study extends the Eclipse Graphical Modeling Framework (GMF) to link graphical editors and modeling technology seamlessly. The unifying force of GMF is the notation meta-model which provides the concrete link between the graphical editor and the data stored and managed using EMF. The most important aspect of the notation meta-model is that it is completely domain independent. By being generic across domains, the engine behaves in a common way and provides a common feature set that can be absorbed into those domains. The only link to the semantic model is a reference stored on the View element in the hierarchy [17]. The core class hierarchy of the notation model is described in Figure 5.9:
Since GMF is designed as a platform for different domain editors to be built on, it needs a robust extensibility solution that will allow clients to extend from other GMF clients and allow their construction to be scalable through the plug-in architecture [17]. A set of services within the GMF diagram plug-ins provides this extensibility for the platform. A service does a specific piece of work for clients by delegating the actual work done to one or more service providers. A service can listen to its provider and request a

**Figure 5.9: Core Class Hierarchy of GMF Notation Model**
provider when it wants to support a given request. Client requests are made using
\textit{IOperation}, the interface for all service provider operations. Each service provider has a
\textit{ProviderPriority} that is declared in its extension descriptor. It is the \textit{ExecutionStrategy}
that determines how service provider priorities are used to select a provider to service
each client request. For example, if the \textit{ExecutionStrategy.FIRST} is used, the provider
with the highest priority will give an answer to the request.

The core services provided by GMF plug-ins are as follows:

- **View Service:**
  ViewService is responsible for setting styles and layout constraints of a notation view
  as appropriate. It adds containment to the view by recursively calling the
  ViewService. A typical service provider returns a factory class that would create a
  notation view that subclasses from one of three abstract classes for creating Nodes,
  Edges or Diagram.

- **EditPart Service:**
  EditPart Service is similar to the View Service except that a provider will return the
  EditPart class that is the controller for the notation view and semantic element.

- **EditPolicy Service:**
  EditPolicy Service is an important extensibility point since it allows clients to install
  their own EditPolicies on existing EditParts without overriding the EditPart
  itself. Each EditPart, when it is created in GMF, will call the EditPolicyService on
  activation to determine if any clients are contributing to their set of EditPolicies.

- **Palette Service:**
The Palette Service is the entry point defining a custom palette for a domain editor with a set of creation tools organized into a drawer or stacks.

- **Decoration Service:**
  The DecorationService is designed as a simple way to annotate an existing shape without knowledge of the implementation of that shape.

- **Layout Service:**
  This serviced will arrange the shapes on the diagram automatically in a more organized manner.
Chapter 6. Modeling of Mobility Services

For any realistic chance of achieving a useful DSM environment, we must thoroughly explore the target domain, develop the architecture behind the product, and form the component library for the product. In this chapter, mobility service is used as an example domain to explain how to build a meta-model that guides the design and development of the mobile services through the composition of the pre-built domain components. A case study is also provided at the end of this chapter to further validate the performance of the proposed framework.

6.1 Mobile Service Overview

Mobile technologies are gaining more popularity and expanding into every aspect of our life. Good mobile services can have a huge impact on our daily life and competition in this field has become critical for service operators. The mobile industry is no longer just about the delivery of voice over phones. The future of mobile telephony is expected to rely on mobile services, and the use of mobile services will be an integral part of the revenues generated by third generation mobile telephony [74]. Introduction of new content services such as voting-contest services, data services, messaging services and Java games makes the mobile industry more dynamic and promising. Non-voice services now represents between 15% and 29% of operator service revenue, depending on the market [75].
Q. Mahmoud, in an on-line article, [76] divides mobile services into two broad categories:

- **Voice and messaging services:** these are basic services available on almost all mobile phones nowadays and are evolving rapidly. The most widely adopted text-messaging protocol is the Short Message Service (SMS), which allows the interchange of short text messages between mobile telephone devices.

- **General applications:** these include personal information management software, mobile front ends to enterprise applications, intriguing games, and many more varieties of mobile software.

We can also divide mobile applications into three models from the view of client-server structure.

- **Local mobile applications**
  These applications execute in a device's own runtime environment. This model supports highly interactive applications, such as intensely graphical mobile computer games. Developers can write directly to each device's proprietary interface through a set of application programming interfaces or APIs. The constraints of mobile devices, on the other hand, also impose challenges to the development of mobile applications and services. Storage capacity, communication bandwidth, processing power, and application interface are some of the typical constraints that limit what can be made available on mobile devices.
Another challenge of developing mobile applications on mobile devices is that mobile developers need to develop for multiple platforms to maximize their available market. This can be a difficult and time-consuming task as multiple platforms use different API calls for common OS operations and for accessing OS resources such as memory and files.

- **Remote mobile applications**

Remote mobile applications run on the servers hosted by mobile network operators or third party mobile service providers, and send the data back to users through the mobile network. Typical use cases can be weather alert, stock tracking, photo manager, and so on. Users normally send requests through SMS with a specific short code. The service provider receives the request, invokes the corresponding service, and sends data back. By moving the processing activities from a memory-limited mobile device to a backend server, this model relieves many of the constraints of local mobile applications. However, new challenges are also introduced by the limited bandwidth and low-rate data transfer.

In past decades, the growth in wireless voice telephony has promoted the development of high-quality, second-generation wireless telephone technology (2G) digital cellular systems. The current 2G systems, such as GSM, provide good quality voice anytime, anywhere. 2G systems provide low-speed data and hence limited data service functionality. With the increasing demand for advanced data services from users, the desire to support high-rate data services, in addition to wireless voice telephony, is
stronger, and it became the main factor in the development of third generation (3G) systems. 3G technologies enable network operators to offer a wider range of more advanced services while achieving greater network capacity through improved spectral efficiency. Services include wide-area wireless voice telephony, video calls, and broadband wireless data, all in a mobile environment. Additional features also include HSPA [83] data transmission capabilities that allow delivering speed up to 14.4Mbit/s on the downlink and 5.8Mbit/s on the uplink. Although 3G is successfully introduced to users across the world, some issues are still debated by 3G providers and users. Some of them are high power usage and the high price of 3G data services in some countries.

- **Distributed mobile applications**

  The distributed computing architecture enables the functional partitioning of applications between devices and servers. It divides the applications in a way that attempts to avoid the constraints of mobile devices, and the limitation of the bandwidth for data transmission. In [84], authors presented experiments that demonstrate the usefulness of compressing web services data both on the server and device side to reduce the web-service message size. The performance impact on the client and server side depends on the message size and connectivity of the testing networks. Limitations and constraints from mobile devices, application servers and wireless networks are the keys for the performance of the distributed mobile applications.
6.2 Mobile Operating Systems

Like a computer operating system, a mobile operating system is the software platform on which other programs run. When you purchase a mobile device, the manufacturer will have chosen the operating system for that specific device. The operating system is responsible for determining the functions and features available on your device, such as thumbwheel, keyboards, WAP, synchronization with applications, e-mail, text messaging and more. The mobile operating system will also determine which third-party applications can be used on your device. In the world of PCs, there's one major OS-Windows and a couple of other operating systems including the Apple Macintosh OS X and Linux. But in mobile devices there are plenty of embedded operating systems including Symbian, RIM, Palm, Windows Mobile, and Linux.

- **RIM Blackberry**: The BlackBerry is a wireless handheld device introduced in 1997 as a two-way pager [86]. The more commonly known smartphone Blackberry, which supports push e-mail, mobile telephone, text messaging, internet faxing, web browsing and other wireless information services was released in 2002. The devices are very popular with some businesses, where they are primarily used to provide e-mail access to roaming employees. RIM provides a proprietary multi-tasking operating system (OS) for the BlackBerry, which makes heavy use of the device's specialized input devices, particularly the scroll wheel (1995 - 2006) or more recently the trackball (September 12th 2006 - Present) [86]. The OS provides support for MIDP 1.0 and WAP 1.2. Previous versions allowed wireless synchronization with Microsoft Exchange Server's e-mail and calendar, as well as with Lotus Domino's e-
mail. The current OS 4 provides a subset of MIDP 2.0, and allows complete wireless activation and synchronization with Exchange's e-mail, calendar, tasks, notes and contacts, and adds support for Novell GroupWise and Lotus Notes.

With BlackBerry deployments, most of the application is sitting on a server, with a small, lightweight client application on the mobile device. Overall, experts agreed that a BlackBerry deployment is optimal for a large-scale deployment in an e-mail-centric business. For smaller deployments or deployments that require access to several thick applications, however, another platform may be needed.

- **Windows Mobile from Microsoft**: Windows Mobile is the Microsoft brand name for the operating system and software applications Microsoft has developed for handheld computers. Devices that run Windows Mobile include Pocket PCs, Smartphones, Portable Media Centers, and on-board computers for certain automobiles [87]. Windows mobile is designed to be similar to desktop versions of Windows, feature-wise and aesthetically. Additionally, third-party software development is available for the Windows Mobile operating system. There are several options for developers to use when deploying a mobile application that includes writing native code with Visual C++, writing managed code that works with the .NET Compact Framework, or Server-side code that can be deployed using Internet Explorer Mobile or a mobile client on the user's device. The .NET Compact Framework is actually a subset of the .NET Framework and hence shares many components with software development on desktop clients, application servers, and
web servers which have the .NET Framework installed, thus integrating networked computing space.

- **Linux Mobile OS**: Linux Mobile OS is the use of a Linux operating system in an embedded computer system. Unlike desktop and server versions of Linux, embedded versions of Linux are designed for devices with relatively limited resources, such as cell phones and set-top boxes. Since embedded devices serve specific rather than general purposes, developers optimize their embedded Linux distributions to target specific hardware configurations and usage situations. Instead of a full suite of desktop software applications, embedded Linux systems often use a small set of free software utilities.

The advantages of embedded Linux over other embedded operating systems include a small footprint, no royalties or licensing fees, a stable kernel, a support base that is not restricted to the employees of a single software company, and the ability to modify and redistribute the source code. Several industry groups have formed to foster use of Linux in embedded applications. The first company that has been involved in launching phones with Linux as OS is Motorola, which brought the first Linux phone to the market in 2003.

Android is a software platform and operating system for mobile devices based on Linux and developed by Google. The unveiling of the Android platform on 5 November 2007 was announced with the founding of the Open Handset Alliance, a consortium of 34 hardware, software, and telecom companies devoted to advancing
open standards for mobile devices [90]. On October 21, 2008, Google released the complete source code for its mobile phone platform with the launch of its Android Market for users to download applications to their Android powered phones. Through Android Markets, application developers are able to make their content available on an open service hosted by Google. Content can debut in the marketplace after only three simple steps: register as a merchant, upload and describe your content and publish it. It provides business opportunities to developers and ultimately improves their offerings.

- **Symbian OS from Symbian Ltd.** [88]: Symbian OS is an open operating system, designed for mobile devices, with associated libraries, user interface frameworks and reference implementations of common tools, produced by Symbian Ltd. On 24 June 2008, the Symbian Foundation was announced with the aim to "provide royalty-free software and accelerate innovation". The APIs are publicly documented and up to Symbian 8.1, anyone could develop software for Symbian OS. The native language of the Symbian OS is C++, although it is not a standard implementation.

- **iPhone OS from Apple Inc.** [89]: iPhone OS is the operating system developed by Apple Inc. for the iPhone and iPod touch. The SDK was released on March 6th, 2008, and allows developers to make applications for the iPhone and iPod touch, as well as test them in an "iPhone simulator". However, different from Google Android, loading an application onto the devices is only possible after paying an iPhone Developer Program fee.
Untill now, developers were free to set any price for their applications to be distributed through the App Store, from which they receive a 70% share. Developers can also opt to release the application for free and not pay any costs to release or distribute the application except for the membership fee. As the iPhone is based on a variant of the same XNU kernel that is found in Mac OS X, the tool chain used for developing on the iPhone is also based on Xcode, which is Apple's suite of tools for developing software on Mac OS X.

6.3 Messaging-based Mobile Services

With more than half the world using mobile phones, mobile messaging is growing at an explosive pace and will continue to do so in the near future. Both entrenched companies and new entrants are providing applications that blur the lines between the traditional mobile messaging applications. New and improved hardware, software, and services are increasing customer choices while bringing down costs. Mobile messages may be used to provide premium rate services to subscribers of a mobile network. For instance, mobile terminated short messages can be used to deliver digital content such as news alerts, financial information, logos and ring tones. Mobile originated short messages may also be used in a premium-rated manner for services such as voting. In this case, the Value-added Service Provider (VASP) obtains a Short Code [81] from the mobile network operator, and subscribers send texts to that number. A value-added service (VAS) is a telecommunications industry term for non-core services or, in short, all services beyond standard voice calls and fax transmissions. On a conceptual level, value-added services add value to the standard service offering, spurring the subscribers to use their
phone more. Mobile phones technologies like SMS, MMS and GPRS are usually considered value-added services. Value-added services are supplied either in-house by the mobile network operators themselves or by a third-party VASP, also known as a content provider (CP) [82]. The subscribers are charged for receiving this premium content, and the amount is typically divided between the mobile network operator and the VASP either through revenue sharing or a fixed transport fee.

Today, the most popular and common used messaging-based mobile services are:

- **Short Message Service (SMS)**: a communications protocol allowing the interchange of short text messages between mobile telephone devices. SMS text messaging is the most widely used data application, with 2.4 billion active users, or 74% of all mobile phone subscribers sending and receiving text messages on their phones [77]. SMS as used on modern handsets was originally defined as part of the GSM series of standards in 1985 as a means of sending messages of up to 160 characters (including spaces), to and from GSM mobile handsets [79]. Since then, support for the service has expanded to include alternative mobile standards such as ANSI CDMA networks and Digital AMPS, as well as satellite and landline networks [77].

- **Multimedia Messaging Service (MMS)**: a new standard in mobile messaging. Like SMS, MMS is a way to send a message from one mobile to another. The difference is that MMS can include not just text, but also sound, images and video. Formats that can be embedded within MMS include:
  
  - Text (formatted with fonts, colors, etc)
- Images (i.e. JPEG, GIF)
- Audio (i.e. MP3, MIDI)
- Video (i.e. MPEG)

MMS is an extension of the SMS protocol, making its usage familiar to existing SMS users. An MMS message is a single entity, not a collection of attachments. One of the main practical differences between MMS and SMS is that while SMS messages are limited to 160 bytes, an MMS message has no size limit and could be many Kbytes in size. MMS requires a third generation (3G) network to enable large messages to be delivered, although smaller messages can be sent even with second generation (2G) networks using GPRS. Possible MMS applications include weather reports, news and sports bulletins, newsletters, etc.

- **Mobile Instant Messaging (MIM):** a presence-enabled messaging service that aims to transpose the Internet desktop messaging experience such as ICQ or MSN to the usage scenario of being connected via a mobile device [92].

In a mobile environment, the user is constrained by bandwidth and the UI. However, the user has the advantage of mobility. MIM users will desire to message with other MIM users as well as fixed IM users on networks such as AOL, Microsoft and Yahoo. MIM allows users to address messages to other users via various bearers using an alias and address book, allowing the sender to know the status of the recipients.
By itself, SMS does not include alias capabilities nor does it allow for confirmation that the intended recipient is available. Being plain SMS does not allow the sender to have a high degree of confidence that the recipient will receive the message in real-time. MIM allows the community of users to register as being present and/or available, allowing for more real-time text messaging and communications than would be possible with traditional mobile messaging.

- **Mobile Email**: Mobile email is the function of composing, reading, replying to, and forwarding emails while on the move. To connect your device wirelessly to your email server, there are many wireless networks you could use. You may require a mixture of wireless networks to achieve what you need such as GPRS, EDGE, 3G, and so on. The most popular technology used by mobile devices to extract email in real-time is push email. Push email utilizes a mail delivery system with real-time capability to “push” email through to the client as soon as it arrives, rather than requiring the client to pull mail manually. With a push email smartphone, for example, the client’s mailbox is constantly updated with arriving email without user intervention. Smartphones announce new mail arrival with an alert.

Push email differs from conventional email systems that are “pull” oriented. Usually, when email is sent, it arrives at the recipient’s Internet Service Provider’s (ISP’s) mail server, where it is held for collection. It might instead arrive at a website server, if the email is Web-based. Either way, email remains on the mail server until the recipient uses an email program to poll the mail server. If new mail is present, the email client “pulls” the mail to the client’s computer. The difference between this scheme and
push email is that, with push email, the mail is pushed through to the client without waiting for polling. Polling involves “handshaking” between the client software and the mail server. If the server is busy, the delay in completing the handshake can lengthen, causing the client to time out. Today, many devices have incorporated push email, and its popularity continues to grow.

6.4 High-level architecture of messaging-based mobile services

Figure 6.1 shows the typical high-level end-to-end architecture of message-based mobile services. The Message Gateway provides the communication mechanism for message exchanges between Mobile Devices and Mobile App Servers hosting applications and services. It enables mobile messaging for mobile applications, such as MMS sending and reception, SMS sending and reception, WAP Push sending, and so on. With Mobile Messaging Gateway, we can add mobile features to existing intranet, extranet or Internet solutions and rapidly develop and deploy new MMS/SMS/push-enabled applications. A Mobile Messaging Gateway can run on the same computer as the applications or on a
The gateway supports multiple applications so SMS/MMS connections can be shared between several users. Applications can send messages to Mobile Messaging Gateway with standard Internet protocols (HTTP). The gateway forwards messages to a suitable MMS/SMS connection according to its routing table.

The function of the Message Dispatcher (see Figure 6.2) is to take messages from the Message Queues and dispatch to the appropriate Mobile App Servers hosting mobile applications and services. These applications and services may interact in turn with legacy systems and database management systems. Billing and accounting systems are some of the typical legacy systems that require integration.

Figure 6.2: Message Dispatcher
A service scenario is illustrated in Figure 6.3. A Mobile Device initiates a service request or invokes a mobile application by sending an SMS message to the Message Gateway over the wireless network. Upon receiving the SMS message, the Message Gateway places the message into a Message Queue. The Message Dispatcher takes the message out of the Message Queue and forwards the message to an application server based on the type of services specified in the content of the SMS message. Services running on the application server interpret the content of the message and perform the necessary functions requested by the mobile device. Information from databases or legacy systems may be accessed to support the message processing. Finally, the application server sends the processing results back to the sending mobile device through the Message Gateway and the result is interpreted by the mobile device.
6.5 Component-based framework for mobile service creation

6.5.1 Related Work

Various types of mobile data applications and services, such as text message services, location-based and presence-based services, multi-media message services, etc., have been proposed, developed, and deployed by wireless service providers and made available to varieties of mobile devices [24] [25]. The development of these applications and services is a challenging task due to the various access technologies, protocols and legacy systems in the wireless network infrastructures. Mobile applications and services need to function in a heterogeneous networking environment and be integrated well into existing authentication, accounting, billing and other legacy systems.

Platform-based and framework-based solutions have been proposed to hide the complexity and heterogeneity of wireless network infrastructures to simplify the creation and deployment of mobile applications and services [26] [27]. These wireless service platforms and frameworks provide a common application programming interface for the development of applications and services independent of wireless communication protocols, network devices, and network resources. By raising the level of design abstraction, such wireless service platforms and frameworks enable rapid service creation and deployment. A programming framework described in [28] provides a generic synchronization model for mobile enterprise applications that use heterogeneous backend stores. A framework for implementing interactive multimedia messaging services is
described in [29] and a programming model is introduced to support the development of these applications. An adaptive middleware infrastructure is described in [30] to simplify the development of mobile applications. The authors in [31] describe an enterprise mobile service platform that delivers end-to-end mobile solutions to large enterprises and simplifies the development of wireless enterprise applications.

Component-based framework solutions have also been used to simplify the development of various domain applications. Authors in [32] describe a component model for safety-critical, real-time, and embedded control systems. A framework for the development and maintenance of component-based software through visualization methods is described in [33]. A component-based generative and model-driven framework in [34] describes the automated software product generation based on the assembly software components.

Nokia uses a commercial metaCASE tool, MetaEdit+ [35], to increase productivity and improve the quality of products. With the encapsulation of domain knowledge, flexible modeling language support and code generation, it gives Nokia great advantage in mobile phones and the mobile computing market. However, the component-based concept is not the focus of its domain-modeling framework. In addition, extending and customizing the existing MetaEdit+ framework to fit with special domain requirements can be complicated and troublesome.
6.5.2 Mobile Service Creation Framework (MSCF)

Operators worldwide are changing from the current methods of delivering services that are neither efficient nor cost-effective and which prevent operators from bundling existing services or introducing new services quickly to a Service Delivery Platform (SDP) based approach. To solve this problem, Motorola announced a GAMA [52] (Global Applications Management Architecture) solution adopting a SDP to provide users with the following functionalities [93]:

- Provisioning of subscribers, applications, and application providers
- Policy management of subscribers, applications, application providers and grouping
- Security features (such as authentication, authorization, privacy, and data integrity)
- OA&M (operations, administration, and maintenance) functionality
- Application deployment, hosting, and execution (decoupling operator from application providers)
• Pre-integrated set of applications that can run seamlessly over GAMA

Even though GAMA SDP enables a set of service level abstractions around telecommunication protocols for application developers to write portable code, it does not provide any application level abstraction – a typical application developer would still have to rely on low-level APIs to code application level business logic. To address this issue, we propose a Mobile Service Creation Framework (MSCF) on top of GAMA SDP, which provides higher-level abstraction and shields developers from the low level APIs (see Figure 6.4).

MSCF adopts a multi-tier architecture for the creation of mobile applications. At the top tier of MSCF, application developers build new mobile applications by means of dragging and dropping pre-built components to a graphical editor. The composition rules and constraints are enforced by the underlying component-based domain meta-model,
which captures the structures and behaviors of mobile services. System configuration and deployment packaging are also specified in the model. The functionalities of the mobile services are mapped to service components in the component library. Integration with legacy systems and data storage is handled by these components. Figure 6.5 shows the high-level view of the framework architecture that uses component-based design and mobile service modeling to specify and implement mobile services. To hide the details and complexity of the wireless networks and communication protocols, the wireless infrastructure is abstracted as components and services with common access interface in this component-based framework to facilitate the development of mobile applications.
Figure 6.5: MSCF Architecture
Mapping the functions of model elements to service components is fairly straightforward. For instance, to send and receive messages, service components for SMS and MMS messages will be used for handling messages of their corresponding types. Parameters for invoking component services are obtained from input and output configuration properties defined in the model. The domain-based design process maintains the mapping rules and associations between service components.

The service creation framework provides a code generation facility to transform an application model into J2EE-based application code. For a generated application, a J2EE enterprise application project is created and the skeleton code of the J2EE application is generated. The code generation facility interprets the application model and its model elements to produce corresponding Java code and J2EE deployment descriptors. The properties of receiving SMS and MMS messages, for instance, are interpreted by the code generation facility to produce code to invoke the service component designed for receiving SMS and MMS messages. The generated code also provides interfaces and extensions for the integration of custom code. The generated application code, hand-written extension and customization code and runtime libraries for service components are packaged as J2EE applications to be deployed on application servers. A browser-based test client is also generated to facilitate the validation of the generated application.

MSCF is implemented as a set of plug-ins in Eclipse, an extensible open source development platform and application framework. The component modeling of mobile services is implemented on the Eclipse Modeling Framework (EMF) and Eclipse
Graphical Modeling Framework (GMF). The applications generated by this framework are packaged and deployed as J2EE applications by leveraging tools from the Eclipse Web Tools Platform (WTP) project for developing and packaging J2EE applications [50]. The packaged mobile applications and services are deployed on the Sun Java System Application Server Platform [51].

6.5.3 Domain Meta-Model of MSCF

As what has been discussed in the previous section, a domain-based design process has been adopted to capture domain knowledge and rules of mobile services and to automate the generation of application code. Different from the previous studies in the domain of mobile service creation, MSCF combines the component-based, modeling, and domain-specific notation to facilitate the design, development, and deployment of mobile services. This section gives more details on how MSCF uses a domain meta-model to specify the structures, behaviors, configuration, and deployment of mobile services. Following the component meta-modeling approach discussed in 5.2, the overall structure of the MSCF meta-model can be formally illustrated in Figure 6.6.
Figure 6.6 MSCF Component Meta-model
The root component *MessagingBasedService* acts as a container for other components to compose a messaging-based mobile service. Every time a message (i.e. SMS or MMS) comes in, an instance of *ServiceServlet*, the servlet component running on the J2EE application servers for message handling, invokes the corresponding *MessageProcessor* component according to the serviceType attribute set in *MessagingBasedService*. The supported services are defined as a type-safe enumeration class *MsgServiceType* in MSCF. Based upon the selected service type, *MessageProcessor* assembles appropriate action components and sends back the composed result to the original sender through the message sending component, *SendMessage*. Furthermore, the component that is responsible for the configuration of the service deployment, *DeploymentDescriptor*, specifies information such as the URL, service types, etc., to facilitate the deployment of the generated mobile application on the J2EE application servers.

MSCF handles web services as a special type of component. *InvokeWebService* component used by *WebServiceInvocation* service invokes corresponding web services in response to a user request.

Web services technology provides a mechanism that facilitates the exchange of data by applications over the Internet. Using open protocols and interfaces, web services technology removes the complexity usually associated with the exchange of data between applications that are running on different devices and using different operating systems. Web services technology lets servers exchange data with other servers, personal computers, and mobile devices.
The technology makes it easy for a client on a mobile device to discover and exchange data with any number of servers. For mobile developers, web services technology can dramatically reduce the complexity involved in creating networked applications. Developed within the Java Community Process as JSR 172, the J2ME Web Services API (WSA) extends the Java 2 Platform, Micro Edition to support web services. The goal of WSA is to integrate fundamental support for web services invocation and XML parsing into a device runtime environment, so that developers don't have to embed such functionality in each application -- an especially expensive proposition in resource-constrained devices like mobile phones and personal digital assistants.

Figure 6.7 illustrates a typical web-service architecture for a WSA enabled device.

![Figure 6.7: J2ME Web Service Architecture](image-url)
At the high-level, this web-service architecture has three elements:

- A network-aware application residing on a WSA-enabled wireless device,
- The wireless network and the Internet, and the corresponding communication and data-encoding protocols, including binary protocols, HTTP, and SOAP/XML,
- A web server, acting as the service producer, typically behind one or more firewalls and a proxy gateway.

Figure 6.8: Web Service Architecture on MSCF
Different from the above approach, MSCF processes all the web-service related activities at the layer of application server, which enables web-service requests to all SMS-enabled mobile devices instead of just to the devices that have WSA support (see Figure 6.8).

In MSCF, *InvokeWebService* invokes corresponding web services from remote systems hosting the requested services in response to the received SMS messages. The result of web services will be forwarded to the end-user in the SMS format.

### 6.5.4 Elements of MSCF Domain Meta-model

After giving an overview of the MSCF domain meta-model structure, in this section we continue with a more detailed description of the key components of the model. This description includes the list of all attributes and interfaces of each component, as well as all aggregation relationships for a certain component, i.e., the definition of what are the allowed children of a certain component and how they are grouped together. Sometimes, for attributes having a limited set of values, we also include the enumeration of the possible values. In addition to the basic description of the elements and attributes, we have attempted to add a more precise explanation, which should motivate the design decisions that have been taken.

- **MessagingBasedService** – The root of any messaging-based mobile service. It represents any mobile service relying on the interchange of web message, short text messages (SMS) or multimedia messages (MMS) that includes multimedia objects, such as images, audio, video and rich text.
Table 6.1: Attributes of MessagingBasedService

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceType</td>
<td>MsgServiceType</td>
<td>An enum type that defines all the possible messaging-based mobile services supported by MSCF. It also provides a custom option for end-users to design their own services.</td>
</tr>
<tr>
<td>ID</td>
<td>int</td>
<td>An integer to uniquely identify mobile services. We have implemented a simple ID generator in Java to generate the necessary key values for each mobile service creation. This is the key attribute, whose value can be referenced by other document elements.</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>The name of the messaging service.</td>
</tr>
</tbody>
</table>

*MESSAGINGBASEDSERVICE* uses a type-safe enumeration class, *MsgServiceType*, to define the available message-based services provided by the framework. A *MESSAGINGBASEDSERVICE* is composed of five different components, *ServiceServlet, DeploymentDescriptor, MessageProcessor, MS_Action*, and *Relationship*. When receiving a SMS, MMS, or web message, an instance of *ServiceServlet* component extracts the message into corresponding *MessageContent* component, and then invokes the relevant *MessageProcessor*, such as *VoteProcessor, TTSProcessor*, and *ReminderProcessor*, based on the value of the serviceType attribute set in *MESSAGINGBASEDSERVICE* component. *MessageProcessor* is the main engine for all the message processing activities. The interface processMessage() is implemented as a call back function in the framework, and based on the selected service type, the corresponding actions are performed in the
sequence defined by the *Relationship* component. Services are finally deployed to the J2EE application server following the description of *DeploymentDescriptor*.

**Table 6.2: Values of the Attribute ServiceType of the Type MsgServiceType**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voting</td>
<td>Typical scenario for this service is that the application is invoked by receiving an incoming message, SMS, MMS or Web Access. The framework extracts the message, delegates the processing to a working component, and then creates an outgoing message (SMS, MMS) based on the output produced by the working component.</td>
</tr>
<tr>
<td>Reminder</td>
<td>Subscribers can use a Web application to enter time, day, etc., to get a reminder service. The system will send the reminder message based on a subscriber’s local time or a specific time zone. Typical examples are birthday reminder (specific time zone), wake-up (local time) call, etc.</td>
</tr>
<tr>
<td>TextToSpeech</td>
<td>Aims to transform arbitrary textual input into spoken output.</td>
</tr>
<tr>
<td>WebServiceInvocation</td>
<td>Allows a mobile application developer to expose web services to users via SMS, such as a web-service that provides flight arrival and/or departure times for a particular airport and flight number. Such a service would be useful in a situation where a person on the way to the airport picks someone up and would like to know if the flight is on time.</td>
</tr>
<tr>
<td>BroadCast</td>
<td>Typical scenario of this service is that administrator configures a database of subscribers to receive a specific message (SMS, MMS) at a particular time, based on a subscriber’s local time. Examples are announcements, group reminder service etc.</td>
</tr>
<tr>
<td>Custom</td>
<td>Gives user the option to create customized mobile applications.</td>
</tr>
</tbody>
</table>
Table 6.3: Content of MessagingBasedService

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceServlet</td>
<td>The framework servlet in response to the applicable events that the application is interested in. The framework provides a Regular Expression based rules engine that allows applications to specify the message types they are interested in. ServiceServlet invokes MessageProcessor according to the received message.</td>
</tr>
<tr>
<td>MessageProcessor</td>
<td>The container component for a list of activity components which are responsible for processing the interchanged web message, test messages or multimedia messages. There is only one interface processMessage() in this component. All the subcomponent of MessageProcessor should implement processMessage() to process the interested message and send back result to the original sender. A messaging service should contain exactly one instance of MessageProcessor component.</td>
</tr>
<tr>
<td>MS_Actions</td>
<td>The container component of all the action components. According to the type of mobile service, different action components are invoked within the processMessage() interface of MessageProcessor. The sequence of the invoked actions is decided by the source and target references of the Relationship component.</td>
</tr>
<tr>
<td>Relationship</td>
<td>The connector between actions. Relationship component contains two references, source and target. An instance of Relationship component will always has two function calls, getSource() and getTarget(), in its implementation, which return source action and target action respectively. This component isn’t designed for direct external use and always goes together with Action components.</td>
</tr>
<tr>
<td>DeploymentDescriptor</td>
<td>Specifies information, such as URL, processor type, etc., to facilitate the deployment of mobile applications on a J2EE application server.</td>
</tr>
</tbody>
</table>
• **ServiceServlet** – A Java Servlet in response to the applicable events that the application is interested in.

In order to receive messages from mobile phone users and deliver the messages to the corresponding `ServiceServlet`, a MessageDrivenBean, `SMSReceiverBean`, is implemented and deployed on the same Java J2EE App Server. Each time a SMS or MMS message arrives, `SMSReceiverBean` will be informed and then trigger the corresponding service servlet on the server according to the content of the message. `SMSReceiverBean` was designed independent of MSCF and, therefore, we will not describe the structure and use of `SMSReceiverBean` in detail.

MSCF provides a regular expression-based rules engine that allows applications to specify the interested message types in `DeploymentDescriptor`. `ServiceServlet` uses the regular expression pattern of the message to bind receiving message with a specific `MessageProcessor` according to the service-mapping table. For instance, if we specify the regular expression as “vote.*” for a particular service, when receiving a message contains the pattern of “vote.*”, `ServiceServlet` will invoke the corresponding `VotingProcessor` to process the message and send back the result. Detailed explanation can be found in the section of MSCF case studies.

**Table 6.4: Content of ServiceServlet**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageContent</td>
<td>Content of the received WEB, SMS or MMS messages. The concrete forms of a <code>MessageContent</code> can be a <code>SMSContent</code>, <code>MMSContent</code>, or <code>WebContent</code>, which contains additional attributes for different type of messages.</td>
</tr>
</tbody>
</table>
MsgServiceContext  Contains service configuration information and servlet related information.

Table 6.5: References of ServiceServlet

<table>
<thead>
<tr>
<th>Name</th>
<th>Target Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>MessageProcessor</td>
<td>Returns an instance of MessageProcessor in response to the received message. Different MessageProcessor type will be returned based on the value of the serviceType attribute of MessagingBasedService. The possible processor can be VoteProcesor, ReminderProcessor, TTSProcessor, BroadcastProcessor, and WebServiceInvocationProcessor.</td>
</tr>
</tbody>
</table>

- **SMSContent**: sub-component of the abstract component MessageContent.

  It contains the information related to a SMS request.

Table 6.6: Attributes of SMSContent

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoneNum</td>
<td>String</td>
<td>The phone number of the user who sent the SMS request. This attribute is programmatically set according to the parsing result of the SMS message, and is not settable by the application developer.</td>
</tr>
</tbody>
</table>
regEx String The regular expression attached with the receiving message. For example, for users who are requesting voting service will have regular expression “vote.*” attached with the sending SMS. This attribute is programmatically set according to the parsing result of the SMS, and is not settable by the application developer.

message String The main content of the text message sent by users through SMS. This attribute is programmatically set according to the parsing result of the SMS message, and is not settable by the application developer.

- **MMSCContent**: sub-component of the abstract component MessageContent. It contains information related to a MMS request.

**Table 6.7: Attributes of MMSCContent**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachment</td>
<td>File[]</td>
<td>An array of the attached files of the receiving MMS. This attribute is programmatically set according to the parsing result of the MMS, and is not settable by the application developer.</td>
</tr>
<tr>
<td>SMIL</td>
<td>File</td>
<td>The SMIL representation of the MMS. SMIL is an XML markup language for describing multimedia presentations. It defines markup for timing, layout, animations, visual transitions, and media embedding, among other things. SMIL allows the presentation of media items such as text, images, video, and audio, as well as links to other SMIL presentations, and files from multiple web servers. This attribute is programmatically set according to the parsing result of the MMS message, and is not settable by the application developer.</td>
</tr>
</tbody>
</table>
o **MsgServiceContext**: its attributes have the servlet related information such as WEB-INFO location and *ServletContext* object of the servlet. It is also the container component of *ServiceConfig* component.

### Table 6.8: Attributes of MsgServiceContext

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webInfo</td>
<td>String</td>
<td>WEB-INF folder location in the file system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The WEB-INF directory is required for a servlet-based web application. It contains a hierarchy in which we can find the necessary configuration information to generate a web application to test mobile services from a web browser and all the class files for the servlets and classes that are called up by the JSPs (Java Server Pages). The testing application will not work without the WEB-INF directory. The value of this attribute is calculated based on the content of <em>DeploymentDescriptor</em> component and not settable by the application developer.</td>
</tr>
<tr>
<td>servletContext</td>
<td>ServletContext</td>
<td>Defines a set of methods that a servlet uses to communicate with its servlet container. Parameter value pairs for ServletContext object are specified in <code>&lt;context-param&gt;</code> tags in web.xml file. The ServletContext parameters are specified for an entire application outside of any particular servlet and are available to all the servlets within that application. This attribute is calculated programmatically at run time.</td>
</tr>
</tbody>
</table>
- **VoteConfig**: sub-component of the abstract *ServiceConfig* component. It contains the configuration of a voting application, such as the format of the input message and how we want to respond the message sender. *VoteProcessor* retrieves the *VoteConfig* data from *ServiceServlet*, and invokes a sequence of *Action* components accordingly.

### Table 6.9: Attributes of VoteConfig

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputFormat</td>
<td>MsgInFormat</td>
<td>An enum type that defines all the supported input formats, such as SMS, MMS, or WEB messages.</td>
</tr>
<tr>
<td>inputItems</td>
<td>List</td>
<td>A list of options for a particular voting application.</td>
</tr>
<tr>
<td>outputFormat</td>
<td>MsgOutFormat</td>
<td>An enum type that defines all the supported output formats, such as SMS_Text, MMS_BarChart, and MMS_PieChart. <em>VoteProcessor</em> uses this value to call the corresponding message sender, i.e., if the output format is MMS_BarChart, <em>VoteProcessor</em> will invoke <em>MMSSender</em>, and display the voting result in the format of a bar chart on sender’s mobile phone.</td>
</tr>
<tr>
<td>outputTitle</td>
<td>String</td>
<td>The title of the resulting page of a particular voting application.</td>
</tr>
</tbody>
</table>
Table 6.10: Values of the InputFormat Attribute of Type MsgInFormat

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS</td>
<td>The request message is in the format of SMS from a mobile device.</td>
</tr>
<tr>
<td>MMS</td>
<td>The request message is in the format of MMS from a mobile device.</td>
</tr>
<tr>
<td>WEB</td>
<td>The request message is an HTTP request from a web client.</td>
</tr>
</tbody>
</table>

- **WSConfig**: sub-component of the abstract `ServiceConfig` component. It returns a list of web services acquired by a particular mobile application.

Table 6.11: Attributes of WSConfig

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>services</td>
<td>List</td>
<td>A list of acquired web services.</td>
</tr>
</tbody>
</table>

- **DeploymentDescriptor**: specifies information to facilitate the deployment of the generated mobile application on a J2EE application server. It contains the configuration information for both the testing and production version of the deployed services, such as the location of the web.xml and service-config.xml, the URL mapping and context root of the testing web application, the regular expression pattern of the service, and so on.
Table 6.12: Attributes of DeploymentDescriptor

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URLMapping</td>
<td>String</td>
<td>The url-pattern associated with the corresponding servlet. When a request arrives, the web server uses a simple procedure for matching the URL in the request with a url-pattern in the web.xml file, i.e., the url for voting related service is “/vote”. The URLMapping is combined with the defined servlet context to compose the full URL to access the servlet. This attribute is required by the web version of the testing service only.</td>
</tr>
<tr>
<td>Container</td>
<td>String</td>
<td>The location where the service’s code generated by the framework is stored.</td>
</tr>
<tr>
<td>context</td>
<td>String</td>
<td>The context root of the web application (WAR). It is only used by the testing browser of the services.</td>
</tr>
<tr>
<td>RegularExpression</td>
<td>String</td>
<td>Regular expression of the service. Servlet uses this value to determine which service has been requested by users. For instance, the regular expression of Voting related service is “vote.<em>”. If user’s SMS message contains “vote.</em>”, VotingServlet will be invoked to handle the request.</td>
</tr>
<tr>
<td>displayName</td>
<td>String</td>
<td>The title that will display on the top of the respond page for a user’s web browser or mobile device.</td>
</tr>
<tr>
<td>configFile</td>
<td>String</td>
<td>The location of the service-config.xml file. It contains service related properties, such as input format, output format, title, and so on.</td>
</tr>
</tbody>
</table>
- **MessageProcessor**: This is the abstract component of all the service processors. All the service processors implement the only interface, processMessage(), of this abstract component. It invokes actions sequentially to process the incoming message and sends a response back with the format defined by ServiceConfig.

### Table 6.13: Interfaces of MessageProcessor

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processMessage</td>
<td>void</td>
<td><strong>content - MessageContent:</strong></td>
<td>All the service processors need to implement this method to process the incoming message accordingly. Based on the regular expression pattern extracted from the received messages, corresponding processors will be activated within a ServiceServlet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>context - ServiceContext:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6.14: References of MessageProcessor

<table>
<thead>
<tr>
<th>Name</th>
<th>Target Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actions</td>
<td>Action</td>
<td>A list of Action components invoked with in processMessage(). Different Action component will be invoked in response to a particular service. For example, OrganizeTask and SendMessage work together for Reminder and Broadcast services, GenerateChart is used by Voting service only in the current framework.</td>
</tr>
</tbody>
</table>

- **ReminderProcessor**: extends abstract component MessageProcessor. It is to process Reminder related messages. Subscribers can use a web application to enter time, day, etc., to get a reminder service. ReminderProcessor sends the reminder message to a user’s mobile device based on a subscriber’s local time or a specific time zone.

Table 6.15: Attributes of ReminderProcessor

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscribers</td>
<td>List</td>
<td>A list of registered mobile phone numbers that subscribe this reminder service. Users subscribe the reminder service via a web browser.</td>
</tr>
</tbody>
</table>

- **BroadcastProcessor**: extends abstract component MessageProcessor. It is to process Broadcast related messages. BroadcastProcessor retrieves a list of service subscribers from the database configured by the system.
administrator and then sends out a specific message (SMS, MMS) at a particular time, based on a subscriber’s local time.

Table 6.16: Attributes of BroadcastProcessor

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscribers</td>
<td>List</td>
<td>A list of mobile phone numbers that subscribe to this broadcast service. Different from Reminder services, subscribers of Broadcast service are defined by a system administrator.</td>
</tr>
</tbody>
</table>

- **VoteProcessor**: extends abstract component *MessageProcessor* to process voting related messages. *VoteProcess* parses incoming mobile/web messages and then saves the user selected vote items into a backend MySQL database. The latest voting result matrix will be calculated and sent back to end-users via *SendMessage* component.

- **TTSProcessor**: extends abstract component *MessageProcessor* to process Text-To-Speech related messages. *TTSProcessor* receives messages from a mobile device or a web client, and transforms arbitrary textual input into spoken output.

- **WebServiceInvocationProcessor**: processes WebServiceInvocation related messages. It exposes web services to end-users via SMS.

- **MS_Actions**: The container component of all the Action components. Different action components are invoked by a particular sub-component of
MessageProcessor within processMessage() with regard to the mobile service type. Some action components are designed for particular services and some can be used across services. For instance, if a voting service is requested, a voting specific action component, GenerateChart, will be called to generate voting results in the format of a bar chart or pie chart. On the other hand, an instance of the shared action component, SendMessage, is initiated for sending the result back to end-users. The order of the actions invoked is defined by the source and target references of Relationship component.

- **GenerateChart**: An action component used by voting services only in the current MSCF. The implementation of this component is independent of MSCF, and thus can be used by other services that require a chart-based response to end-users. The type of the desired chart is defined in VoteConfig component. GenerateChart retrieves chart type through the “content” parameter of the processMessage() interface.
### Table 6.17: Attributes of GenerateChart

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>Color</td>
<td>The color of the generated chart diagram.</td>
</tr>
<tr>
<td>seriesColors</td>
<td>Map</td>
<td>A list of key and value pair objects. The key of the object refers to the name of a voting item from the option list. The value refers to the representation color of a voting item.</td>
</tr>
<tr>
<td>chartType</td>
<td>ChartType</td>
<td>A enum type that provides a list of chart types. The current supported types are BarChart and PieChart.</td>
</tr>
</tbody>
</table>

The value of this attribute is retrieved via the “content” parameter of the processMessage() interface, and not settable by application developers.

### Table 6.18: Interfaces of GenerateChart

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generateChart</td>
<td>Image:</td>
<td>SelectedItems (SMSContent):</td>
<td>This operation calculates the current voting result and returns an image in the form of a graphical chart.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The chart diagram of the voting result.</td>
<td>The received SMS message that can be used to calculate the current voting result.</td>
</tr>
</tbody>
</table>
- **SendMessage**: An action component to send SMS or MMS to a designated mobile device, or send web content to a web client.

## Table 6.19: Interfaces of SendMessage

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendSMS</td>
<td>boolean:</td>
<td></td>
<td>This operation sends the text message to the phone number defined in parameter Msisdn.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Msisdn (String):</td>
<td>the telephone number to the SIM card in a mobile/cellular phone.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message (String):</td>
<td>The text of the return message.</td>
</tr>
<tr>
<td>sendMMS</td>
<td>boolean:</td>
<td></td>
<td>This operation sends the image message to the phone number defined in parameter Msisdn.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Msisdn (String):</td>
<td>the telephone number to the SIM card in a mobile/cellular phone.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message (Image):</td>
<td>The Image file of the return message.</td>
</tr>
<tr>
<td>sendToWeb</td>
<td>boolean:</td>
<td></td>
<td>This operation sends a text message or a graphical chart to the client’s web browser. It is only used by the web-based testing tool of the mobile services.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>response (HTTPServletResponse):</td>
<td>the object to assist a servlet in sending a response to the client’s web browser.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message (Object):</td>
<td>the message that needs to be sent to the client’s web browser. It can be either a text message or a graphical chart.</td>
</tr>
</tbody>
</table>


- **ConvertTextToSpeech**: an action component to convert normal language text into a WAV speech file.

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processTTS</td>
<td><strong>boolean</strong>: the flag indicates whether or not the operation has been performed successfully.</td>
<td><strong>message (String)</strong>: The text that will be converted into speech. The speech file generated is in WAV format. <strong>Context (String)</strong>: The location in which the generated speech file is saved.</td>
<td>This operation converts normal language text into Waveform audio format (WAV).</td>
</tr>
</tbody>
</table>

- **OrganizeTask**: an action component to schedule tasks for repeated execution by specifying a fixed rate of execution or a fixed delay between executions. This action is used by services like Reminder and Broadcast.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Unique identifier of the task.</td>
</tr>
<tr>
<td>startTime</td>
<td>Date</td>
<td>First time at which task is to be executed.</td>
</tr>
<tr>
<td>interval</td>
<td>long</td>
<td>Time in milliseconds between successive task executions.</td>
</tr>
<tr>
<td>occurrences</td>
<td>int</td>
<td>The number of times the task repeats.</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>The text content of the task to the end-users.</td>
</tr>
<tr>
<td>Name</td>
<td>Return Type</td>
<td>Parameters</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>create</td>
<td><code>boolean:</code></td>
<td>none</td>
</tr>
<tr>
<td>edit</td>
<td><code>boolean</code></td>
<td>none</td>
</tr>
<tr>
<td>delete</td>
<td><code>boolean</code></td>
<td><code>Subscribers (List):</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ID (String):</code></td>
</tr>
</tbody>
</table>

- **InvokeWebService**: An action component to invoke web services from remote systems hosting the requested services in response to the received SMS messages. The result of the web services will be forwarded to the end-user in the format of SMS.
Table 6.23: Interfaces of InvokeWebService

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invokeService</td>
<td>result (String): message (String): wsc (WebServiceConfig):</td>
<td>an SMS text message representing the response of the acquired web services. message (String): the SMS request from a mobile device. wsc (WebServiceConfig): the WebServiceConfig object extracted from MessageProcessor.</td>
<td>This operation invokes web services according to the received SMS. Web service results are returned to service requester in the format of SMS initially, which can be extended to support both SMS and MMS return types in the future.</td>
</tr>
</tbody>
</table>

- **Relationship**: A Relationship component is interpreted as a directed edge linking two actions in the model diagram. An instance of Relationship cannot exist without both of the actions that it links. This component depends on the existence of the connected Action components and thus should be used within the context of MSCF.

Table 6.24: References of Relationship

<table>
<thead>
<tr>
<th>Name</th>
<th>Target Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Action</td>
<td>The source action, from which the Relationship edge begins.</td>
</tr>
<tr>
<td>target</td>
<td>Action</td>
<td>The destination action, pointed to by the Relationship edge.</td>
</tr>
</tbody>
</table>
6.5.5 MSCF Visual Editor

In addition to the component domain model, a graphical editor based on the technology of Eclipse GMF has been provided for composing messaging-based mobile services, where the model components and pre-built components in the component library are assembled following the rules defined in the domain model. As discussed in section 4.4.4 for building a GMF editor, we need to define a number of additional models on top of the domain model defined in the previous section:

a) a notation model (.gmfgraph) that defines the graphical notation including shapes, decorations and graphical nodes and connections.

b) a tool definition model (.gmftool) for the editor's palette

c) and a mapping model (.gmfmap) that binds the notation model and tooling model to the domain metamodel.

• The Graph Model of MSCF Editor

The graphical definition model defines a set of figures in the default FigureGallery. Colors, line widths and static decorations of the figures are also defined here. We also define graph nodes and connections in the graph model where each node or connection maps to the respective figure definition in the FigureGallery. Compartments are sections in nodes that can be collapsed and themselves contain graphs. Finally, we define diagram labels used to show text associated with graphical elements.
Figure 6.9 shows that all the graph nodes named after the domain model elements we want to map in the mapping model of the MSCF editor. In the property page of each node (which is not displayed in Figure 6.9), we connect the node elements to the respective figures. For example, the *Deployment* node is associated with the *DeploymentFigure* in the graph model.
Figure 6.9: Graph Model of MSCF Editor
• The Tool’s Definition Model of MSCF Editor

The tool definition model (Figure 6.10) defines only a set of palette entries in MSCF. The palette is the set of elements on the right of the MSCF editor that allows application developers to add model elements to the graphical editor. Therefore, we need a creation

Figure 6.10: Tool’s Definition Model of MSCF
tool for each of the domain model elements that we want to be able to place in the editor.

We also assign icons to each of the creation tools. Note that we don't need a creation
tool for the root element, `MessagingBasedService`, since the `MessagingBasedService` will
be represented by the editor as a graphic canvas.

- **The Mapping Model of MSCF Editor**

![Mapping Model of MSCF](image)

Figure 6.11: Mapping Model of MSCF
This is the crucial and most complex model for the creation of MSCF editor. In this mapping model, we map the tool definition and the graph definition to the domain model in property pages. As highlighted in Figure 6.11, node “Actions” in the graph model has been mapped to “Creation Tool Actions” in the tool model and “MS_Action” component in the domain model.

- MSCF Editor

Figure 6.12 shows the running editor with an example mobile service in the editor.
The editor assembles appropriate components and implements the service by mapping the model elements to service components and generating application code from the model. The tool palette of the editor shows the available components for the service creation. Table 6.25 shows the mapping between the palette elements and the represented components. Every component is associated with a property page that controls behavior of the corresponding model object. It allows a user to communicate with the components and change the behavior of the components. Domain rules and constraints are enforced by the framework during the service composition in the editor.

Table 6.25: Elements Mapping of MSCF Editor

<table>
<thead>
<tr>
<th>Palette Element Icon</th>
<th>Palette Element Name</th>
<th>Represented Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Start Icon]</td>
<td>Start</td>
<td>VoteServlet</td>
</tr>
<tr>
<td>![Deploy Icon]</td>
<td>Deploy</td>
<td>DeploymentDescriptor</td>
</tr>
<tr>
<td>![VoteProcessor Icon]</td>
<td>VoteProcessor</td>
<td>VoteProcessor</td>
</tr>
<tr>
<td>![BroadcastProcessor Icon]</td>
<td>BroadcastProcessor</td>
<td>BroadcastProcessor</td>
</tr>
<tr>
<td>![ReminderProcessor Icon]</td>
<td>ReminderProcessor</td>
<td>ReminderProcessor</td>
</tr>
<tr>
<td>![TTSProcessor Icon]</td>
<td>TTSProcessor</td>
<td>TTSProcessor</td>
</tr>
<tr>
<td>![Actions Icon]</td>
<td>Actions</td>
<td>Action</td>
</tr>
<tr>
<td>![SendMessage Icon]</td>
<td>SendMessage</td>
<td>SendMessage</td>
</tr>
<tr>
<td>![TextToSpeech Icon]</td>
<td>TextToSpeech</td>
<td>ConvertTextToSpeech</td>
</tr>
<tr>
<td>![OrganizeTask Icon]</td>
<td>OrganizeTask</td>
<td>OrganizeTask</td>
</tr>
<tr>
<td>![PieChart Icon]</td>
<td>PieChart</td>
<td>GenerateChart</td>
</tr>
</tbody>
</table>
6.6. MSCF Case Study – Voting Service

6.6.1 Overview

One of the most popular cultures spawned by the exponential growth of the mobile phone is mobile voting. Contests such as American Idol utilize mobile voting to make the show more interactive. The process of mobile voting usually involves keying in a certain keyword which corresponds to a certain category or individual.

If properly utilized, mobile voting can prove to be a very beneficial tool for business’ success and growth. It allows companies to anticipate current market trends so that they can adapt properly to reap the benefits of current trends. Mobile voting could play a crucial role on monitoring how the products or services are doing in the market. Data gathered in this manner could then be used to formulate the proper business move or strategies to improve the market acceptance of products and services.
Companies could also make use of mobile voting services to garner customer feedback on a product or event that they wish to launch. Using mobile voting services, companies get a wider target population, who can express their support and suggestions using their mobile phones. This makes mobile voting services a cost-effective tool to assess the market strength of the product or event. Mobile voting could be a more innovative and appealing suggestion box for the target market and the regular client base.

6.6.2 Development Usage Scenario

![Figure 6.13: A Sample Voting Application](image)
As displayed in Figure 6.13, voters of a particular event send a voting request to a predefined short code (an universal five digit code to identify an application) location either via a web page or a mobile phone. We collaborated with network operators to provision the wireless network to route SMS/MMS messages to the Motorola GAMA platform. The corresponding voting service deployed on MSCF will be invoked by the incoming SMS/MMS message. An instance of VoteServlet component parses the message and delegates the processing to VoteProcessor component. GenerateChart and SendMessage components will then customize the response to automatically include live vote results in the format of a chart diagram. The diagram (Figure 6.14) below describes the sequence of events for a voting message roundtrip application.

Figure 6.14: Voting Sequence Diagram
New voting applications can be rapidly created by composing the corresponding components within the visual MSCF editor. Properties of components are settable through the property page of each component provided by the framework. The steps involved in composing, configuring, packaging and deploying a voting application are described as following:

1. Application developers create a dynamic web project using the Eclipse Web Tools platform. This web project acts as a container for the files generated by MSCF.

2. Application developers launch the MSCF editor. MSCF editor allows developers to choose components from the palette and compose them in a visual editor.

3. The editor canvas maps to the root component, `MessagingBasedService`, of mobile services. To build Voting service, developers need to set the `serviceType` attribute of `MessagingBasedService` to “Voting”.

4. Add the “Start” element from the palette to the editor. “Start” element will automatically map to “VoteServlet” component in MSCF according to the `serviceType` value set in step 3.

5. Add the VoteProcessor to the editor. Since the service type is set to “Voting” in step 3, only “VoteProcessor” element is accessible from palette.

6. Add “Actions” to the editor, which acts as a compartment of all the voting related action components.
7. To show the voting result in a pie chart, Drag the “BarChart” element from palette, and drop it into “Actions” compartment. GenerateChart component will be invoked to draw a bar chart to display the current voting result.

8. Drag the “SendMessage” element from palette, and drop it into “Actions” compartment after “BarChart” element. The “SendMessage” component will send the pie chart to end-users’ mobile phone or web browser depending on how users send their request.

9. Add the “Deploy” element to the end of this application flow. The “DeploymentDescriptor” component corresponds to this diagram element and developers can set the deployment configuration via the property page of this component. Figure 6.15 shows the model instance generated following the above
10. Right click any empty place of the editor to open the context menu, choose “Build Mobile Service” to generate all the required files to the web project created in step 1.

11. To package the application, developers can create a war file using the war creation wizard provided by the Eclipse Web Tools Platform.

12. Deploy the .war file on GAMA application server. When users send voting message through the mobile network, network operator will forward the message to GAMA server according to the short code embedded in the message. GAMA server forwards the message to the voting application based on the regular expression pattern contained in the message and then send the voting result back to message senders. Figure 6.16 shows a sample voting result as a bar chart on a Motorola Razr phone.

![Figure 6.16: Voting Result on a Mobile Phone](image-url)

![Motorola Razr Phone with voting result](image-url)
6.6.3 Conclusion:

The world is changing rapidly with the introduction of newer and more powerful mobile devices. People rely on mobile devices and the provided services in their daily life. With this reliance on mobile services comes the challenge that new applications are desired or old applications must be re-written and constantly maintained soon after they are deployed to keep up with the pace of changing technology.

To respond to the rapidly changing technology, the proposed framework provides a visual graphical interface with the underlying constructs linking to the supported component built on top of Motorola GAMA platform. It accelerates the creation of mobile applications and is easier for non-experts to master mobile application development. Development errors are also greatly reduced as a result of the use of pre-built domain components. From our case studies, over 80% of the application code can be generated from model-based design specifications. The performance and code size of these generated applications are comparable to those created without design automation. The development of mobile applications and services can be automated or significantly simplified based on our limited case studies.
Part II

Domain-Specific Modeling for a Dependable Component-based Software System
Chapter 7. A Model-Driven Approach to Implementing Dependable Component-Based Mobile Services

7.1 Overview

Various design methodologies and architectural paradigms have been proposed to tackle the complexity and reliability of large-scale software systems. Both software components and web services are often used to reduce design complexity. Software components and web services serve as large building blocks, and the construction of large software systems integrates and assembles these software building blocks. They provide flexible and scalable solutions for the design and integration of complex systems and applications [3, 53, 54].

The construction of a software system with component composition brings new challenges to the dependability of the system. Software components, including commercial off-the-shelf (COTS) components, offer various degrees of dependability, depending on their functional complexity, implementation, and deployment environment. Their dependability may be unknown during the design phase or vary significantly depending on the platforms on which they run. The composition and interactions of
components also contribute to the overall system dependability and further complicate system analysis. While some of the components may be hosted in the same execution environment as their applications, the deployment of web-service components is independent of their consumers. Hence, the availability and reliability of a web-service component is beyond the control of its client applications. For systems that require high dependability, the failures of components and services must be handled to minimize their impacts on the overall availability and reliability of the systems.

Authors in [14] describe a source-to-source compiler technique that applies source code transformation rules to introduce code modification for fault detection. Data and code duplication was introduced to improve software dependability. Authors in [15] introduce a multiple-view modeling approach to ensure component dependability. Component models are specified with four modeling perspectives: component interface, static behavior, dynamic behavior, and interactions. Consistencies and dependencies among the models are established and maintained to achieve system dependability. In [16], a self-healing mechanism is proposed to build reliable systems based on connectors. Besides synchronizing message communication between tasks in a component, connectors are extended to support self-healing by detecting anomalies in anomalous objects, reconfiguring objects in components, and repairing the anomalous objects detected.

The framework described in this study takes the integrated modeling of application domains, components, web services, and their dependability. The framework also automates the generation and integration of both application code and a failure
protection code from the domain models. Compared to previous approaches, the integrated framework described in this study allows the specification of component and service dependability at the model level independent of languages and platforms. Platform and language-specific code is generated from models to implement both the functional and dependability specifications.

For clarification, the term dependability means both system availability and reliability throughout this paper. In some literature, dependability may be used as a collective term for reliability, availability, maintenance, safety, and security.

7.2 Component with dependability extensions

The proposed framework takes a model-driven approach to integrate components and represent them as model elements. Their functions and relationships with other modeling or programming constructs are specified at the model level. The modeling of components is further extended with the specification of their dependability.

In this study, web services are treated as a special type of component that focus on machine-to-machine interaction over a network. Web services frequently are just web APIs that can be accessed over a network, such as the Internet, and can be executed from remote systems hosting the requested services. To improve interoperability of Web Services, a set of core specifications with some additional requirements have been published to improve interoperability of web services. In the meantime, companies like BEA Systems, IBM, Microsoft, etc, worked closely to propose a specification protocol allowing SOAP messages to be delivered reliably between distributed applications in the
presence of software component, system, or network failures. The resulting WS-ReliableMessaging 1.1[94] was approved as an OASIS (Organization for the Advancement of Structured Information Standards) standard on June 14th, 2007.

Because of the particular characteristics and unreliable infrastructure of web services, we adopt different strategies to handle dependability of regular components and web-service components. Figure 7.1 shows the representation of components as model elements and their associations with a dependability specification.

Both ServiceComponent and WebService are specializations of component Service. They are modeled and validated based on their interfaces for composition. Components and services are replaceable as long as they are compatible at the interface level. OCL constraints can also be attached to the meta-models to capture more syntactic and semantic information for component composition, enabling stronger correctness

![Figure 7.1. Modeling Component Dependability](image-url)
checking beyond the syntax checking and semantic checking of the traditional component-based software development [5, 6]. Connectors linking components and services enforce the integration rules and constraints. With the formal specification of components and web services as model elements, their functional specification and interactions can be verified within the context of a domain model. One of the implications for such protection mechanisms is that it requires the protected software units to be stateless to ensure that failure handling is transparent to the application logic. Since software components and web services are generally defined to be stateless, their failures can be handled by the above protection mechanism without impacting the application logic. For stateful software units, the failure handling needs to be customized.

The goal of modeling components and web services along with their dependability specifications is to automatically generate the protection mechanisms based on the handling strategies declared in the dependency specification. The dependability specification of a component or a web-service is implemented by a delegation pattern with extension to handle the failures of component and service invocations. Invocations of components or web services are delegated to their concrete implementations, and failures from invocations are intercepted, interpreted, and handled according to the strategies in the dependability specification [39].

Figure 7.2 extends the MSCF with the dependability specification with which components and web services are associated.
Components and web services are associated with component DependabilitySpec to specify their failures and corresponding handling mechanisms. The dependability specification for MessageProcessor and Actions is specified by ComponentDepSpec. WSDepSpec is associated with web-service related Action components, such as InvokeWebService. The dependability specification together with the functional specification of components and web services provides complete information to generate the protection code and associate it with the invocation of components and web services.
7.3 Elements of Component Dependability Extension

The key elements composing the specification of component dependability in Figure 7.1 are explained as following:

- **ServiceComponent** - the specialization of Service Component. It represents all the regular components used in the MSCF domain model.
- **DependabilitySpec** - an abstract component to extend the specification of components and web services to incorporate the specification of component and service failures and the strategies to handle these failures. Different types of failures and their associated handling strategies are described in ComponentDepSpec and WSDepSpec. Failure types include both value-based and exception-based types. For value-based failures, the return values from the component and service invocation indicate the types of failures. For example, the return value of “-1” may be used to indicate a specific failure of component invocation, similar to the conventions used in some of the UNIX library calls. Exception-based failure types represent the types of exceptions declared in and thrown by components and web services. An exception is any error condition or unexpected behavior encountered by an executing program. Exceptions can occur due to a number of reasons such as a fault in application code, an operating system resource not being available, an unexpected condition in the common language runtime, and so on. While applications can recover from some of these conditions, most of the runtime exceptions are irrecoverable. In that case, we
need an effective way of handling those exceptions and informing the callers of the exception.

In Java code, exception handling is provided by try...catch...finally statement. The try keyword precedes a block of normal processing code that may throw an exception. The catch keyword precedes a block of exception handling code. The finally keyword precedes a block of code that will always be executed after handling the exceptions. Once an exception is thrown from a try block, the program flow switches to the first catch block following it. A well-designed set of error handling code blocks can go a long way in making the program more robust and less prone to crashing because of the way the application handles such errors.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failureType</td>
<td>FailureType</td>
<td>An enum type that defines the format of a return failure from a running component. It includes ErrorFailure and ExceptionFailure.</td>
</tr>
<tr>
<td>exceptionType</td>
<td>Exception</td>
<td>The types of exceptions declared in and thrown by components and web services</td>
</tr>
<tr>
<td>errValue</td>
<td>int</td>
<td>Return values from the component and service invocation indicate the types of failures.</td>
</tr>
</tbody>
</table>

Component DependabilitySpec also defines the strategies to control how each of the failure handlers are triggered. For example, the control mechanism may specify the
number of component failures before a component will be restarted. It may also specify how redundant web services are invoked in the presence of web-service failures.

- **ComponentDepSpec** – extends abstract component *DependabilitySpec*. It contains *ComponentFailureHandler* to handle different types of failures return by the initiated components.

- **ComponentFailureHandler** – provides a set of strategies to handle different failures returned by a component. The following types of handlers can be defined for components in response to the failure value returned: re-invoking the component service, restarting the component, and ignoring component failure.

**Table 7.2: Interfaces of ComponentFailureHandler**

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handleFailure</td>
<td>void</td>
<td><strong>failureType</strong>: FailureType</td>
<td>Component handler may reinvoke, restart, or ignore the running component in response to the different component failure type. Maximum retry times has been defined as an attribute, MAX_RETRY, in <em>ComponentFailureHandler</em>. Component failures will be ignored once the number of re-invoke/re-start component service exceeds the value of MAX_RETRY.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>failureValue</strong>: Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>The type of the component failure. It can be either value-based or exception-based failure.</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>The value of the failure type. It can be an integer for value-based failure or a String type for exception-based failure.</strong></td>
<td></td>
</tr>
</tbody>
</table>
Table 7.3: Attributes of ComponentFailureHandler

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_RETRY</td>
<td>static int</td>
<td>The maximum time for a component failure handler to retry the component service.</td>
</tr>
</tbody>
</table>

- **WSDepSpec & WSFailureHandler**: The failures of web services are handled in a similar fashion with the exception that there may be multiple service providers for a particular web service. Component *WSDepSpec* defines a list of web-service providers [95] and allows the invocation of a web-service from any available web-service provider. In the event of a web-service failure, the contained *WSFailureHandler* defines a set of rules in terms of how failures should be handled, such as invoking the list of web-service providers sequentially until the invocation succeeds or the list depletes.

### 7.4 Generating dependable applications

The dependability specification dictates how component failures are handled in the generated code. Figure 7.3 illustrates the scheme of handling component failures. A “Dependable Delegate” is generated to handle all interactions with a component. The “Dependable Delegate” defines all the interfaces declared in the component. An
invocation (such as “invokeService()”) is passed to the delegate which in turn routes it to the component to invoke the actual service. Invocation results are returned to the invoking application via the delegate. For the application, the function of the inserted delegate is transparent and the overhead is minimal since an invocation simply passes through the delegate if the invocation is successful. If the invocation fails, the “Dependable Delegate” will act upon the failure based on the failure handling strategies declared in the dependability specification. Actions may include re-invoking the service, restarting the component, or simply passing the failure along. The handling of web-service failures follows a similar scheme.

As an example to illustrate the generation of failure handling code, the invocation of logging a call detail record without failure handling, billingComp.log(aCDR), is transformed to the following code template to handle component failure:

```java
if (depSpec.isValueErrType()) {
    //call billing service to log call-detail-record (aCDR)
    result = billingCompProtector.log(aCDR);
    if (result == depSpec.getErrValue()) {
```

![Figure 7.3: Generated Failure Handler for Components](image)
Different types of failure handling strategies in the ComponentFailureHandler and WSFailureHandler are used to handle different types of components and web services based on the strategy specification. Web services, for instance, may require the handling of connection time out. If the same web-service is deployed on two hosts, one of the handling strategies is to pick one host to invoke the service first and then try the other one if the invocation fails. The failure handling strategies are automatically translated into implementation code and integrated with the application code.

7.5 Building and deploying dependable message-based mobile services

As discussed in chapter 6, the model-based component framework was employed to develop and pilot a number of mobile applications and services for several wireless service providers. We collaborated with these wireless service providers to provision the
wireless networks to route SMS and MMS messages and support the deployment of mobile applications that are designed and generated from our model-based component framework. Results from the model specification, generation, and deployment of mobile applications and services showed substantial improvement in software development productivity over our own code-centric approach.

Figure 7.4 illustrates a message-based mobile application that checks flight status and provides weather information for mobile users. A mobile station sends an SMS message with flight numbers and carrier names to the service provider (predefined SMS code). The application extracts the content from the message, invokes the flight status web service, composes an MMS message containing flight status with advertisement, and sends the composed MSM message back to the sender. The usage of the service is also logged.

**Figure 7.4: A Dependable Message-Based Mobile Application Model**
An instance of *WSDepSpec* (dependability specification for web services) is created and associated with *FlightStatusService*. Redundant web services (multiple web-service hosts offering the same service) are specified using this instance and they are configured to be invoked in a round-robin fashion. Similarly, an instance of *ComponentDepSpec* is associated with logging services to ensure usages are recorded reliably.

### 7.6 Summary

This chapter describes a model-based approach for the specification and generation of dependable mobile applications and services. The key contributions of this chapter are the integrated specification of functions and dependability in domain-specific models, the automatic generation of protection mechanisms to minimize component and web-service failures, and the generation of applications from domain models to reduce coding efforts. The pilots of this framework with applications developers and service providers showed productivity improvement in developing and deploying dependable mobile services.

Dependability specification is an integrated part of the MSCF domain meta-model to generate domain applications with improved availability and reliability. Compared to our previous code-centric approach to developing mobile applications and services, substantial improvement on productivity and dependability was achieved with the integrated framework and its development environment.
Integrating components and web services into model-driven development can be a very effective solution to improving both software productivity and quality. With the specification of component and service dependability, further improvement in software quality and reliability can be achieved.
Chapter 8. A Model-Driven Approach to Implementing Component-Based Real-Time Mobile Services

8.1 Overview

Another key factor of application dependability, the construction and deployment of real-time mobile services, faces a number of challenges, especially for the service-based invocation of distributed web services. The loosely-coupled nature and independence of web services invoked in mobile applications make it difficult to achieve real-time assurance since there is little control over their runtime performance. Another challenge is the constraints on the wireless networks where airtime, bandwidth, and processing power are key cost factors. The architectural decision to provide real-time assurance should also take cost factors into consideration.

Integration with legacy systems further complicates the design and deployment of real-time mobile services. The business process specifications must provide a means to integrate existing legacy service components, such as accounting, authentication, and authorization, which do not conform to web services.
The majority of the work on real-time mobile services has been focused on voice-based services and mobile video services and little work has been done on WS-BPEL related mobile data services. A real-time service-oriented architecture was proposed in [57] providing real-time service modeling, design, code generation, simulation, deployment, execution, orchestration, and management. An algorithm was proposed for optimal composition of services with real-time constraints. A methodology for managing real-time enterprise was proposed in [58] for service-oriented business performance management. Performance goals of real-time enterprise are achieved through service monitoring, analysis, and the measurement of key performance indicators.

The authors in [59] describe mobile services that enable the real-time collection and processing of information residing in a community of mobile terminals and authors in [60] describe an SOA-based architecture to support real-time exchange of information between heterogeneous systems for emergency response systems. WIP (Web Service Initiation Protocol) [96, 97] is a web-service based communication paradigm for real-time communication and converged communication services over IP. As a full-featured web-service protocol, each WIP endpoint is implemented as a web service. In [98], WIP is further integrated with the Android mobile platform, and therefore makes an Android-based mobile terminal also a web services endpoint which can be registered, discovered, integrated, and protected through standard-based web services methods. Authors in [99] proposed a set of enabling services as web services for the billing of context-aware real-time mobile services. A questionnaire-based service configuration tool was implemented
to allow a non-expert service provider to easily configure and deploy context-aware mobile services on an open SOA platform.

The SIRENA project [100] leverages SOA to seamlessly interconnect devices from different domains such as industrial, telecommunication, automotive, and home automation. The key component of the SIRENA framework, Device Profile for Web Services (DPWS), arranges several web services specifications such as WS-Addressing, WS-Discovery, WS-MqttadatExchange, and WS-Eventing. As claimed in [78], SIRENA can provide real-time functionalities through a real-time IP stack that interoperates with the Quality of Service (QoS) aspect of SIRENA. IP Multimedia Subsystem (IMS) [40] introduces an efficient architectural framework for delivering rich multimedia services to mobile users, especially for real-time, interactive network based mobile gaming. Different from traditional user equipment based service interaction, IMS handles service interaction such as incoming phone calls and messaging, during a real-time mobile game session at the network level. With the proposed model in [40], mobile devices do not need to have any intelligence about the game or service interaction, which makes all IMS-based games device independent.

Our approach differs from previous work in a number of ways. We provide a model-based approach to specifying and implementing real-time assurance for mobile services through distributed deployment and monitoring of processes and process activities. With domain notations, the construction of real-time mobile services can be simplified. The distributed execution of mobile services facilitates the optimization of resources and real-time assurance for deploying time-critical applications and services in
a wireless access environment where resources can be significantly constrained. The
modeling approach provides a higher level of abstraction than WS-BPEL itself and enables application developers to specify real-time constraints, deployment configuration, and the integration of legacy components, systems, and services in a consistent manner.

### 8.2 Modeling time constraints

Figure 8.1 illustrates a portion of the meta-model for specifying components, web services, legacy services, and real-time constraints for mobile services. A mobile service is specified as an instance of a BusinessProcess, which consists of a number of activities. Specific activities include WebService, LegacyService, Flow, Sequence, and other types of activities. To support real-time mobile services, an activity or a group of related activities (an instance of flow or sequence) can be associated with the specification of real-time constraints using the RT_spec modeling construct.

The construct LegacyService in the mobile service meta-model is a container for integrating with legacy systems and services, such as accounting services, location-based services, etc. Its purpose is to enable the interactions with legacy systems in the same way as web services for the construction and execution of mobile services specified using the extended WS-BPEL.

The Deployment_spec construct is designed specifically to optimize the deployment of mobile services on distributed service architectures to improve performance and meet real-time constraints. An activity, a group of activities, or a sub-process can be specified to execute on mobile clients where resources are very limited or on servers with more resources and processing power. As discussed earlier, airtime,
bandwidth, and other resources can be significant cost factors for wireless access networks. Distributing the processing of mobile services in both the clients and servers can be an effective approach to resource optimization and real-time assurance. In some cases, an activity can only be executed on servers, due to the constraints of legacy systems. Details of the distributed mobile service architecture and runtime support for remote deployment are discussed in the next section.

The real-time specification of an activity specifies the duration of the activity and the actions to be taken if the execution of the business activity exceeds the specified duration. Different types of handlers can be attached to handle real-time violations of business activities. For hard real-time constraints, the handler may report the failure of the business process. For soft real-time constraints, on the other hand, the handler may perform an alternative operation, depending on the specification in the business process.

Figure 8.2: Distributed Mobile Service Architecture
8.3 Distributed mobile services architecture and runtime monitoring of mobile services

Figure 8.2 shows the high-level mobile service architecture we experimented with for deploying mobile services in a GSM network. In the previous work and pilots with network operators, all mobile service logic is hosted in the Mobile Service App Servers (MSAS) and access to mobile services by Mobile Clients has to go through MSAS (from dependable above). In this architecture, both the Mobile Clients and the MSAS share the execution of mobile services (business processes in WS-BPEL). An activity or a group of activities can be executed on either the client or the server depending on the Deployment Spec discussed in previous section.

The distributed deployment of mobile services is designed to optimize their performance by taking into consideration the different characteristics in the Wireless Access Network and its external network (Internet). In general, the Wireless Access Network offers more predictable real-time communication between Mobile Clients and the MSAS, but is more limited in bandwidth. Web service invocations outside on the Wireless Access Network, on the other hand, are more abundant in bandwidth, but less predictable in performance. Our goal is to improve response time by reducing the amount of traffic to and from the Mobile Clients and moving time-consuming computations to the App Servers.

A typical mobile service scenario is as follows. Here, we assume the mobile service specification is already deployed on the server and it is also available at the
mobile client. The execution of an activity may be done at the mobile client or the App Server depending on the specification of the activity. For an activity to be performed on the App Server, the mobile client instructs the App Server to do so and waits for the response back from the server. The process proceeds to complete the execution of activities or aborts when exceptions occur.

We implemented a runtime system to support both distributed execution of

![Figure 8.3. Runtime Support for Real-time Services](image)

Figure 8.3. Runtime Support for Real-time Services
activities and monitoring of these activities to ensure that the real-time constraints are not violated. For activities or sub-processes with remote deployment specifications, the request for remote execution is initiated by the mobile client and the results of the server side execution are sent back to the client. Real-time monitoring of process execution is implemented at both the client and server to monitor the execution of activities or sub-processes with real-time constraints to ensure that real-time activities or sub-processes are completed within the specified time duration or violations will be reported.

Figure 8.3 illustrates the scheme of distributed real-time execution of a mobile service. Before the execution of the business process, the process specification is loaded onto both the client and the server. In this illustration, the business process consists of a sequence of five activities (activities 1 to 5). Activities 2 and 3 are specified to be run on the server side. At runtime, these two activities are replaced with an activity proxy on the client side and the actual execution of activities 2 and 3 is delegated to the server. Results from the server are sent back to the proxy and the execution of the process at the client side continues.

Real-time assurance is implemented by monitoring processes at both the client and the server. As shown in Figure 8.3, the execution of activities 1 to 5 is required to run to complete within T1 and activity 3 within duration T2. As activity 1 starts, a timer for T1 is started on the client side to monitor the execution sequence of activities 1 to 5. When activity 2 completes and activity 3 starts, a timer for T2 is also started on the server side to monitor the execution of activity 3. Since activity 3 completes within the specified time, the timer for T2 is reset and results are sent back to the client and the server.
completes its execution for the specified process. The client process continues to complete activity 4. As this point, the timer for T1 expires, the real-time violation is reported, and the client process is informed to abort its execution.

8.4 Building and deploying mobile-based group communication services

To verify the feasibility of the proposed method, we developed a group communication service to provide emergency notifications based on locations and tasks on our mobile service platform. Currently, mobile-based group communication systems are based on “flat lists” (same connectivity to all users) that are defined either statically (e.g. using Instant Message groups) or dynamically (e.g. using Selective Dynamic Group Call available from some of the mobile terminals). For example, multiple teams of various specialties may be put together to complete a large construction project and team members may rely on mobile communication systems to coordinate. Team members in such an environment may join and leave the group depending on their tasks and locations. We extend the current group communication schemes to add real-time constraints.

While most of these group-based mobile applications do not have hard real-time constraints, many of them require soft real-time assurance. For some applications, however, real-time communication can be a crucial factor. Teams of people working as a collaborative unit in a hazardous and life-threatening environment, for instance, must have the means to communicate among team members immediately or within a required
time framework under critical situations to ensure the safety of the team members involved.

We proposed a mobile-based group communication service to enable such real-time notification using messages. This service is specified as a WS-BPEL process with annotation added to specify (1) real-time constraints on the activities, flows, and sequences, and (2) the options to execute them on either mobile clients or servers hosting mobile messaging services. In this group notification service, we assume that all mobile users are pre-registered in the system in order to be notified. A member of the registered mobile users can initiate an emergency notification (consisting of a textual message and a time limit) to be sent to all the other registered users. If the real-time constraints are not satisfied, the mobile that initiates the notification will automatically call emergency

![Figure 8.4: A Mobile-based Group Communication Service](image-url)
service 911 for immediate assistance. For special situations, such as a large construction site with no fixed communication infrastructure, this type of notification service can be a useful alternative to alert people involved.

Figure 8.4 illustrates the process workflow of the mobile-based emergency notification service. As mentioned earlier, all the mobile users are pre-registered to receive emergency notifications and the process activities related to registration are not shown in this diagram. When an urgent situation arises, a mobile user will initiate emergency notifications and the process execution starts with the process activity “Initiate Notification MSG”. Once the server receives the notification message, it populates a list of current on-site subscribers from the result of the T2 block in Figure 8.4. The notification message will be sent to all the subscribers and the notification process must be completed within duration T1. The server can optionally update the notification message to add location maps showing all emergency exits, emergency procedures to be followed, etc. To ensure all the registered mobile subscribers are notified, the notification process will wait for acknowledgement from all the registered mobiles after the notification is sent. If the notification and acknowledgement processes cannot be completed within the time frame T1, it resorts to the emergency service 911 (simulated action) as an alternative. In the scenario where one or more mobile subscribers are missing or fail to acknowledge, the application deployed on the server will calculate the location data of the missing members referring to the existing site map and create an overlay to quickly locate the missing members. The new map will be sent with other messages to notify the remaining subscribers so that actions can be taken by nearby group
members. The handling of both exceptions and real-time violations is implemented uniformly based on our framework of dependable software systems.

The activities within the T2 block in the diagram run in a separate thread to optimize the notification process by integrating with location-based services. The server periodically queries the location-based service and uses the location information to determine whether a notification is necessary for the client depending on its location (e.g. a team member in a construction team moves in or out of the notification zone). In addition, the location information of the missing group members is a key factor for organizing more timely and precise rescue plans.

As shown in Figure 8.4, the highlighted process activities are executed on the clients and the other activities on the server. We can see from the above example that distributed workflow execution facilitates real-time assurance of mobile services. Time-consuming tasks and the exchange or processing of volume data, such as broadcasting a multimedia message to a large number of users, should be kept to the server to minimize the time needed to complete the tasks. Optimization, however, doesn’t guarantee real-time performance. The execution of mobile services must be monitored to ensure they complete within the specified time frame. In our experiment, separate threads are created to monitor the execution of processes. Furthermore, a balanced use of synchronous and asynchronous messaging is critical to ensuring near real-time performance of mobile services. The initiation of the emergency service, for instance, is time critical and thus is a synchronous web services invocation, whereas the notification and acknowledgement processes that although are time critical, have less stringent time constraint and therefore
can use asynchronous SMS. On the other hand, web services invocation from a mobile terminal requires the mobile terminal to be GPRS-enabled in the context of GSM, which is not available for most of the mobile phone users in contrast to SMS. Our framework requires only SMS communication from the registered subscribers to the server that makes the service available to everyone who has an SMS-enabled mobile device. Only the people who are responsible for the initiation of the service need a mobile device with GPRS access.

8.5 Summary

This chapter describes a model-based approach to leveraging and extending WS-BPEL process specification for the development of real-time mobile services. The extended process model can be used to specify distributed deployment of processes with real-time constraints and integration with legacy systems to support the development, deployment, and monitoring of real-time mobile services. The key contributions of this work are the distributed approach to implementing real-time assurance for mobile services, the model-based design to enable the specification of real-time constraints and integration of legacy systems, and the generation of deployable mobile services through model transformation.

This work is still preliminary. While the distributed execution of activities and sub-processes facilitates the realization of real-time assurance, it highly depends on the deployment environment and capabilities in mobile terminals and servers.
Chapter 9. Conclusion

Traditional approaches to making mobile applications take a lot of expertise, time, and coding. To address these challenges and enable users, not just seasoned developers, to extend applications and processes to mobile users, this study demonstrates a new mobile service creation framework leveraging technologies from different areas, such as component-based software engineering, and domain-specific modeling. An Eclipse based graphical modeling tool was developed to deliver an ease of use design and development environment for users to create their own mobile business scenarios by means of drag and drop. The second part of this study extends the proposed mobile application creation framework to address the approaches to building dependable services. The prototypes of dependability and real-time specifications have been proposed and integrated to MSCF with little impact to the existing system. Several proof-of-concept pilots have been conducted using the proposed framework, and the result of the framework is very well received and acknowledged. From our case studies, over 80% of the application code can be generated from model-based design specifications. The performance and code size of these generated applications are comparable to those created without design automation. The development of mobile applications and services can be automated or significantly simplified based on our limited case studies.
Future research will focus on advancing the current framework with enhanced debugging and testing support. To assist model-driven development, we must provide an environment that allows model debugging and execution. The use of executable models enables the visualization and discovery of defects early in the development cycle, thus preventing costly rework at later stages. Eclipse SDK provides a debug platform for building and integrating debuggers. It defines a set of Java interfaces that model standard debugging functionalities, including the ability to set breakpoints, step executions, and suspend, resume, and terminate threads. The debug platform is robust in such a way that other programming languages can use the debug facilities for their respective language runtimes. Further investigation needs to be conducted on the integration of the Eclipse debug framework with MSCF.

I will also need to work more closely with domain experts to refine and enrich the domain element collection. Trade-offs between domain-specificity and general-purpose programming language constructs need to be balanced so that users can easily create new domain applications from existing components.
References:


[10] P. Nordwall, TheServerSide,

http://www.theserverside.com/tt/articles/content/LightweightModeling/article.html


http://www.eclipse.org/modeling/

[16] Eclipse Foundation, Eclipse Modeling Framework (EMF),

http://www.eclipse.org/emf/.


[36] Microsoft Domain-specific Language tools,

http://msdn.microsoft.com/vstudio/DSLTools/


[40] M. Balakrishnan and M. Sadasivan, “Mobile interactive game interworking in IMS”, 


[43] Visual programming language, 


http://www.eclipse.org/webtools/.

[51] Sun Microsystems, “Java System Application Server Platform”,

[52] Motorola Global Applications Management Architecture (GAMA),
http://www.motorola.com/content/0,,20378181,00.html.


http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html


156


[85] Developer Guide to the GMF Runtime Framework,


