MODELING AND CODE GENERATION FOR A REFERENCE SOFTWARE ARCHITECTURE FOR NAVAL PLATFORM COMMAND AND CONTROL SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NAFİYE KÜBRA TURHAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2017

Approval of the thesis:

MODELING AND CODE GENERATION FOR A REFERENCE SOFTWARE ARCHITECTURE FOR NAVAL PLATFORM COMMAND AND CONTROL SYSTEMS

submitted by **NAFİYE KÜBRA TURHAN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _______________

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _______________

Prof. Dr. Halit Oğuztüzün
Supervisor, **Computer Engineering Department, METU** _______________

Examining Committee Members:

Prof. Dr. Ahmet Coşar
Computer Engineering Department, METU _______________

Prof. Dr. Halit Oğuztüzün
Computer Engineering Department, METU _______________

Prof. Dr. Ferda Nur Alpaslan
Computer Engineering Department, METU _______________

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Department, METU _______________

Assist. Prof. Dr. Hacer Yalım Keleş
Computer Engineering Department, Ankara University _______________

Date: _______________

ABSTRACT

MODELING AND CODE GENERATION FOR A REFERENCE SOFTWARE ARCHITECTURE FOR NAVAL PLATFORM COMMAND AND CONTROL SYSTEMS

Turhan, Nafiye Kübra
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Halit Oğuztüzün

September 2017, 104 pages

Many software teams who work in a particular domain develop software products compliant with a specific Reference Software Architecture. By adopting a Reference Software Architecture within an organization, software development schedule tend to shorten, efficiency of software development process and quality of software product tend to increase.

Architectures of all application software that are developed by Sea Defense Systems Software Team are created based on a predefined Reference Software Architecture named Sea Defense Systems Reference Software Architecture (DSS-RSA). In this thesis, we propose a Model Driven Engineering approach to enforce the Reference Software Architecture and to facilitate the process of transition from architectural design of application software to implementation. In this approach, we create a metamodel for describing DSS-RSA. Then, we define a domain specific graphical modeling language based on the metamodel. In the last stage, models that are created by using the domain specific graphical modeling language are automatically transformed to skeleton code. The approach has been evaluated on a case study.

Keywords: Model Driven Engineering, Reference Software Architecture, Domain Specific Graphical Modeling Language, Automatic Code Generation, Model to Text Transformation

ÖZ

DENİZ PLATFORMU KOMUTA KONTROL SİSTEMLERİ REFERANS YAZILIM MİMARİSİ İÇİN MODELLEME VE KOD ÜRETİMİ

Turhan, Nafiye Kübra
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Halit Oğuztüzün

Eylül 2017 , 104 sayfa

Belli bir alanda çalışan birçok yazılım ekibi özel bir Referans Yazılım Mimarisi ile uyumlu yazılım ürünleri geliştirmektedir. Bir organizasyon içinde özel bir Referans Yazılım Mimarisinin benimsenmesi sonucunda, yazılım geliştirme takvimi kısalma eğilimi gösterirken, yazılım geliştirme sürecinin etkinliği ve yazılım ürünlerinin kalitesi artma eğilimindedir.

Deniz Savunma Sistemleri Yazılım Ekibi tarafından geliştirilen tüm uygulama yazılımlarının mimarileri, Deniz Savunma Sistemleri Referans Yazılım Mimarisi (DSS-RSA) olarak adlandırılan önceden tanımlanmış bir Referans Yazılım Mimarisine dayanarak oluşturulmuştur. Bu tezde, uygulama yazılımlarının mimari tasarımından, kodlanmasına geçiş sürecini kolaylaştırmak ve yazılımların Referans Yazılım Mimarisi ile uyumlu olarak geliştirilmesini sağlamak için bir Model Güdümlü Mühendislik yaklaşımı önerilmektedir. Bu yaklaşımda, DSS-RSA'yı tanımlamak için bir metamodel oluşturulmuştur. Ardından, metamodele dayanan alana özgü bir grafiksel modelleme dili tanımlanmıştır. Son aşamada, alana özgü grafiksel modelleme dili kullanılarak oluşturulan modeller otomatik olarak iskelet koda dönüştürülmüştür. Önerilen yaklaşım bir vaka çalışması üzerinde değerlendirilmiştir.

Anahtar Kelimeler: Model Güdümlü Mühendislik, Referans Yazılım Mimarisi, Alana Özgü Grafiksel Modelleme Dili, Otomatik Kod Üretme, Modelden Metine Dönüştürme

To my lovely family

ACKNOWLEDGMENTS

I would first like to thank and express my gratitude to my thesis advisor Prof. Dr. Halit Oğuztüzün for his encouragement, supervision and guidance throughout the research.

I also would like to thank ASELSAN A.Ş. for giving me the opportunity of continuing my education. I wish to express sincere appreciation to my colleagues and seniors in my department for their support.

I wish to thank to my supportive friends and all people who have helped and inspired me during my thesis study.

Finally, I have no suitable word to express my deepest gratitude to my family for their support in every aspect of my life. I am indebted to my mother Emine Turhan and my father Mustafa Turhan for their care and everlasting love to me. My sister Fatmanur Turhan and my brother Hasan Turhan thanks for their love, trust, understanding and every kind of support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ATL	Atlas Transformation Language
C2	Command and Control
DSL	Domain Specific Language
DSS	Sea Defense Systems
DSS-RSA	Sea Defense Systems Reference Software Architecture
DSS-Factory	Sea Defense Systems Software Development Framework Generator
EMF	Eclipse Modeling Framework
GMF	Graphical Modeling Framework
GUI	Graphical User Interface
HTML	Hypertext Markup Language
JET	Java Emitter Templates
JS	JavaScript
KKYTM	Command and Control Software Design Department
MDE	Model Driven Engineering
MOF	Meta Object Facility
MVC	Model-View-Controller
MVC-RIA	Model-View-Controller Rich Internet Application
M2T	Model to Text
oAW	openArchitectureWare
OMG	Object Management Group
PDE	Plug-in Development Environment
OCL	Object Constraint Language
PIMM	Platform Independent Metamodel

PSMM	Platform Specific Metamodel
RIA	Rich Internet Application
RSA	Reference Software Architecture
SMC	State Machine Compiler
SQL	Structured Query Language
SST	Defense System Technologies
UML	Unified Modeling Language
UWA	Ubiquitous Web Application
WST	Web Standard Tools

CHAPTER 1

INTRODUCTION

As the structural complexity and size of software systems increase, the need for improving productivity of software development process in terms of quality, time and cost also increases. It is difficult to deliver a software product that works correctly and efficiently in a limited time. Recently, software engineering world overcome this challenge by strengthening the efficiency of software development process. Since software architecture is the backbone of a successful software and the key element that determines the quality of a software, it is being thoroughly investigated. To achieve quality aspects of software, the research area of software architecture has grown up and has accumulated important knowledge [4]. In the light of the obtained knowledge, Reference Software Architectures started to be used in software systems to develop better quality software products in a fast and effective way. By adopting a Reference Software Architecture within an organization, software development schedule tend to shorten, efficiency of software development process and quality of software product tend to increase [5].

Reference Software Architecture is a type of generic software architecture. It accumulates the founding principles, underlying methodology and the architectural practices that are recognized by the domain experts as the best solution [6]. Reference Software Architectures comprise a family of software architectures for a specific domain. Concrete software architectures are instantiated from Reference Software Architectures. It provides standardized and systematic reuse of knowledge, components and core assets for the development of a concrete software architecture for a particular software product [7], [8].

However defining a Reference Software Architecture is not enough. Software teams who create particular architectural designs compliant with a predefined Reference Software Architecture spend a big portion of their time on implementation. Further, developers must ensure the architecture compliance of a software product. In some cases architecture erosion problem occurs. The architecture erosion problem is defined as the discrepancy between the architecture description and the resulting implementation [9]. Software systems can change over time: bug fixes can be done or new features can be added. Such changes can result in architecture erosion problem. Furthermore, implementation of software may diverge from original architectural design due to short development deadlines or inexperienced developers in the architectural design. In such cases, it becomes difficult to further develop, maintain or understand the code and advantage of well designed architecture is lost. Inconsistencies emerged at the beginning of the implementation have great importance. Making the implementation compliant with the software architecture becomes hard and costly due to the divergence of implementation from architectural design. In some cases these problems can lead to re-implementing the complete system or at the end of the software development lifecycle, erroneous, costly and poor quality software product may be delivered to customer [10].

Sea Defense Systems (DSS) Software Team in Command and Control Software Design Department (KKYTM) of Defense System Technologies (SST) division in ASELSAN Inc. creates architectural designs of all application software in compliance with a predefined Reference Software Architecture named as Sea Defense Systems Reference Software Architecture (DSS-RSA). In this thesis, we propose a Model Driven Engineering approach to enforce the Reference Software Architecture and to facilitate the process of transition from architectural design of application software to implementation. We describe benefits of our approach. Preliminary results of this thesis has been presented in [11].

1.1 Motivation

DSS Software Team develops software in the fields of above-water and underwater platforms of weapon/sensor control systems, decision support systems, sonar systems, fire control systems and other naval command and control (C2) systems and components. They use a predefined Reference Software Architecture for their architectural designs, namely DSS Reference Software Architecture (DSS-RSA). DSS-RSA is defined by DSS Software Team as a common architectural structure to support the development of individual software products. But developers spend a lot of time on implementation after architectural design is done. Occasionally, there may be cases where the software architectural design of individual applications is non-compliant with Reference Software Architecture or it is compliant with Reference Software Architecture structurally, but implementation of software does not follow the architectural guidelines. For these reasons, developers of DSS Software Team need to perform architectural design of a new project in compliance with the Reference Software Architecture, carry out implementation compliant with the architectural design and minimize the time spent for implementation.

To ensure Reference Software Architecture compliance of architectural design, prevent architecture erosion problem and shorten development time, we put forth a Model Driven Engineering approach. In this approach we:

- Develop a metamodel of Reference Software Architecture put forth by DSS Software Team (DSS-RSA)
- Define Object Constraint Language (OCL) constraints for DSS-RSA Metamodel
- Define graphical concrete syntax based on the DSS-RSA Metamodel and develop a graphical modeling editor
- Create models by using graphical modeling editor
- Develop a code generation mechanism using Xpand template language
- Generate skeleton codes by using code generation mechanism and models

1.2 Contributions

The main contributions of this thesis are as follows:

- A Model Driven Engineering approach to automate the transition process from architectural design which is compliant with the predefined Reference Software Architecture to implementation for the DSS Team Software Projects is devised.

- A graphical tool is developed in order to enable the users to do architectural designs in compliance with the predefined Reference Software Architecture, namely DSS-RSA. In other words, Reference Software Architecture compliance of software architecture is promoted.

- Through automatic code generation, architecture erosion problem is largely prevented and architecture compliance of software implementation is promoted.

- Time spent to develop the software is reduced significantly through automatic generation of software framework.

- The quality of the delivered software is enhanced because there will be no errors resulting from non-compliance to the Reference Software Architecture.

1.3 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 presents the related work on Model Driven Engineering techniques, Reference Architectures and Reference Architecture Metamodeling.

Chapter 3 describes the Reference Software Architecture put forth by DSS Software Team and its layers. In addition, a mechanism for software framework generation used by DSS Software Team is presented in the rest of Chapter 3.

In Chapter 4, metamodeling concept, metamodeling language concept, DSS-RSA Metamodel and mapping from DSS-RSA to Metamodel are described.

In Chapter 5, OCLinEcore Editor and defined Object Constraint Language (OCL) constraints for DSS-RSA Metamodel are explained in detail.

Chapter 6 presents the graphical concrete syntax definition based on the DSS-RSA Metamodel.

In Chapter 7, the code generation mechanism using Xpand template language is described.

In Chapter 8, conducted case study and evaluation of results are presented.

In Chapter 9, discussion and conclusion are presented. The work to be done in the future is mentioned.

CHAPTER 2

RELATED WORK

Reference Software Architecture, Model Driven Engineering and Automatic Code Generation are important topics in Software Engineering community and many research studies have been published on these topics in recent years. In this chapter, we summarize some of these recent works that are related to the study we described in this thesis. In Section 2.1, the studies which use Model Driven Engineering tecniques are mentioned. In Section 2.2, a study that propose correspondence between Reference Architectures and Metamodels is described. In Section 2.3, studies on Reference Architecture Metamodeling are mentioned.

2.1 Model Driven Engineering

Trojanek [12] apply a Model Driven Engineering approach to design domain-specific solutions for subsumption based robot control system development. The study in [12] identifies the modeling concepts, creates subsumption control architecture metamodel and defines OCL expressions. Additionally, in the same study, a graphical notation editor using Eclipse Modeling Framework (EMF) Project is developed to allow the designer to specify the structure of the control system. After the model definition with graphical notation editor, about 1800 lines of skeleton Ada code of the robot control system application are generated automatically by applying model to code transformation with Eclipse Acceleo.

Altunbay et al. [13] describe a Model Driven Engineering approach in order to increase the productivity of computer games design and development. [13] introduce a metamodel using Meta Object Facility (MOF) metalanguage and extension mechanism of the Unified Modeling Language (UML) metamodel and provide a Domain Specific Language for board game domain. In the same study, static sematic rules via OCL expressions are defined and two example board game model (Chess game model and Backgammon game model) are provided. The study in [13] does model to model transformation via Atlas Transformation Language from Board Game metamodel to Video Game metamodel and model to text transformations from Chess game model to Java source code by applying three different methods via MOFScript tool which is an implementation of the MOFScript model to text transformation language, via OpenArchitectureWare (oAW) tool and via Xpand language.

In [14], the authors propose a Model Driven Engineering approach for fast prototyping of Rich Internet Applications (RIAs). They use well known Model Driven Engineering frameworks and technologies including Eclipse EMF, GMF and Xpand2. The authors define two metamodels, the first of which enables designing a RIA at a conceptual level using the Ubiquitous Web Application (UWA) design methodology and the other one is a Model-View-Controller RIA (MVC-RIA) metamodel adopting the Model-View-Controller (MVC) architectural design pattern and RIA widgets for the user interface. In the same study, the authors develop a UWA Graphical Model Editor to define the conceptual model and use Atlas Transformation Language (ATL) transformation rules to automatically transform the UWA conceptual model into the MVC-RIA design model. MVC-RIA design model can be customized and refined using a MVC Graphical Model Editor. After the design refinement step, developers do model to text transformation via Xpand to generate source code (HTML/JS resources, Java source code, SQL scripts, and project metadata) of a ready-to-deploy prototype of the application that uses the RichFaces Framework. The authors also develop a case study to validate the proposed approach and design and implement an e-commerce RIA named e-Market. e-Market application is ready to be deployed on a Tomcat 7.0 application server using a MySQL database for data persistency.

Saritas and Kardas [15] propose a Model Driven Engineering approach that ease the development of smart card software. They define a Platform Independent Metamodel (PIMM) for smart card systems and two smart card Platform Specific Metamodels (PSMM) which are Java Card and ZeitControl Basic Card metamodels. Platform independent and platform specific metamodels are defined by using Ecore metamodel included in the Eclipse Modeling Framework. Model constraints are implemented with OCL. To model the smart card elements and their relationships graphically, they develop modeling editors for both platform specific and platform independent metamodels by using Eclipse Graphical Modeling Framework (GMF). Developers design smart card models according to the platform independent metamodel by using graphical modeling editor. Then these models are transformed into the models of smart card execution platform such as JCF or Basic Card environment (model-to-model transformation between instances of platform independent metamodel and platform specific metamodel). They use ATL for model transformations. They perform model-to-text transformations between platform specific metamodel instances obtained from model-to-model transformation and software code by using the MOFScript. Developers can also design smart card models in the platform specific level by using platform specific graphical modeling editors and use these models as the direct input for model-to-text transformation to generate program code in Java and ZC-Basic Languages. In the end, they realize the same smart card system on different execution platforms and provide easy development of smart card software saving the developers from the tedious and error-prone work.

Eloumri [16] does a case study by using GMF for the creation of a graphical diagram editor. This graphical diagram editor is developed for State Machine Compilers (SMC). Domain model of SMC is created as EMF Ecore Model. Graphical concrete syntax is developed by using Eclipse GMF. SMC model instances created by the graphical diagram editor is transformed into SMC source code. Model to text transformation is done by using the Java Emitter Templates (JET) which is a part of Eclipse M2T project. Strengths and weaknesses of GMF observed during the case study are listed.

2.2 Correspondence between Reference Architectures and Metamodels

Graciano Neto et al. [7] carry out a Systematic Literature Review that shows there is a need for advances regarding Reference Architecture representation and tools to manipulate models. [7] claim that metamodels and Reference Architectures are correspondent in a conceptual level and Model Driven Engineering techniques, frameworks, processes and methods can also be used in Reference Architecture management. They compare characteristics of metamodels and Reference Architectures, both are abstract models that concrete models can be derived from. Both are seperated in views. Specific diagrams are used to model their views and there is a need for both to check the conformance between the concrete and the abstract model.

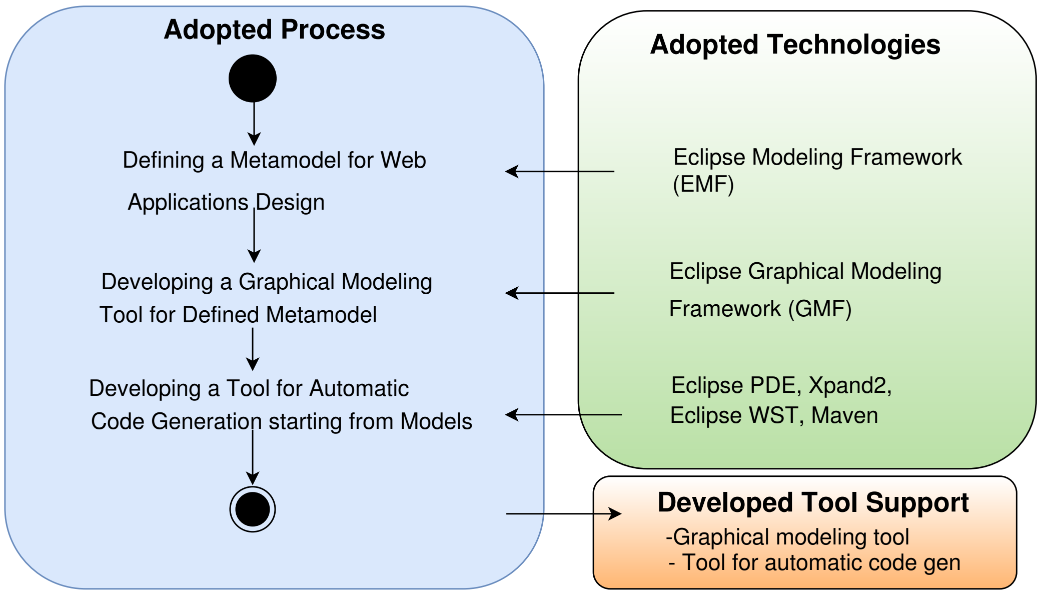
2.3 Reference Architecture Metamodeling

There are some studies that create metamodel of a Reference Architecture and apply Model Driven Engineering techniques for software development, for example [17] and [1].

Miksovic and Zimmermann [17] propose a domain-specific decision knowledge processing solution to cover the requirements of complex strategic outsourcing solutions design. Firstly they determine a set of architecturally significant requirements. Then, they developed conceptual reference architecture from the requirements. Finally, they design a decision process-oriented metamodel by using the reference architecture. This metamodel defines a Domain Specific Language (DSL) as a workflow language that provides certain concepts which allow to model and configure a decision guidance system. They also generated a tool that allows knowledge engineers to model detailed design variations and the relationships between them. In the end, solution design decisions can be managed effectively and they become comparable by detecting deviations from standards and best practices.

Bernardi et al. [1] propose an approach for the model driven fast prototyping of Web Applications in order to reduce the risk of rework during software development and increase the code reusability and quality. [1] chooses a reference architecture along with MVC architectural design pattern and JavaServer Faces technology platform for Web Application development. [1] creates the metamodel of reference architecture and a graphical editor is developed to create, view, and edit models which are instantiated from the defined metamodel. Then, [1] defines model to text transformation rules to automatically transform models defined using the developed graphical editor into source code conforming to the chosen reference architecture and platform. In

Figure 2.1 adopted process and technologies and developed tool support by [1]] is shown. To define the metamodel the Eclipse EMF, to develop the graphical editor



Adopted Process
Defining a Metamodel for Web Applications Design
Developing a Graphical Modeling Tool for Defined Metamodel
Developing a Tool for Automatic Code Generation starting from Models
Adopted Technologies
Eclipse Modeling Framework (EMF)
Eclipse Graphical Modeling Framework (GMF)
Eclipse PDE, Xpand2, Eclipse WST, Maven
Developed Tool Support
-Graphical modeling tool
- Tool for automatic code gen

Figure 2.1: Process and technologies adopted by [1] to define model driven fast prototyping approach and resulting tool support

Eclipse GMF, to validate correctness of the generated models, OCL is used by [1]. [1] chooses the Xpand template language for code generation (HTML/JS resources, Java source code, SQL scripts and project metadata), because of its excellent support of EMF metamodels. [1] develops a case study that automatically generate the prototype of a simple Web Application for on-line note taking and sharing. The resulting Web Application is ready to be deployed on a servlet container like Apache Tomcat, using MySQL as Database Management System for data storage. [1] generates fully functioning prototype of Web Application automatically and rapidly. In the end of the study, design refinement process is simplified, verifying and validating the design is made possible.

CHAPTER 3

DSS REFERENCE SOFTWARE ARCHITECTURE AND SOFTWARE FRAMEWORK GENERATOR

DSS Software Team develops software products using Reference Software Architecture in the fields of surface and underwater platforms, weapon/sensor control systems, decision support systems, sonar systems, fire control systems and so on.

In this chapter, DSS Reference Software Architecture and Software Framework Generator is described. Firstly, in Section 3.1, Reference Software Architecture put forth by DSS Software Team and its layers are given in detail. In Section 3.2, a mechanism for software framework generation used by DSS Software Team is presented.

3.1 DSS Reference Software Architecture

DSS Reference Software Architecture (DSS-RSA) is defined as a common architecture to support the development of individual software products. DSS-RSA involves high-level definitions that are going to be used in software development. DSS-RSA is a general architecture and can be used by different domains. DSS-RSA is defined by adapting an architecture that Microsoft offers to .NET developers. This widely used and trusted architecture helps developers build effective, high quality applications more quickly and with less risk [18].

Architectures of all software that are developed by DSS Software Team are created based on this Reference Architecture. DSS-RSA offers a layered architecture and directives for the new software to be compliant with this layered structure. Figure 3.1 shows conceptual architecture definition of DSS-RSA.

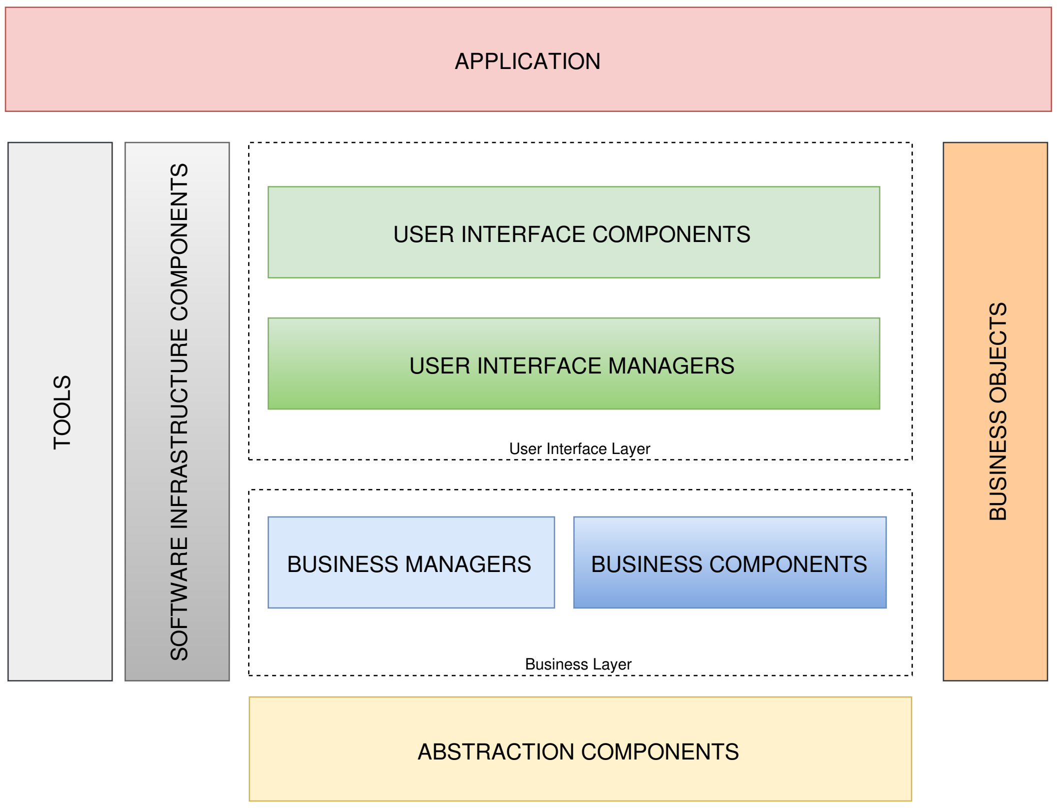


Figure 3.1: DSS Reference Software Architecture layers

In Figure 3.1, the boxes in layers are called as Component. However, this should not be construed as developing each of them as a component, but it would be more appropriate to consider it as a piece of software. Descriptions of the components in Figure 3.1 and their relationships with each other are given below:
Tools: These components involve software units that perform simple computational functions that may be needed by all software components like unit conversion tools, text formatting tools, object comparison/hashing tools, text operations tools.

Software Infrastructure Components: The basic function of Software Infrastructure Components is the Software Lifecycle Management. This component group meets the requirements of other layers like localization management, data store management etc.

User Interface Components: These components involve software units that implement Graphical User Interface (GUI) classes. There are no functionality capabilities in these components. Management of user interface functionality is handled by User

Interface Managers. User Interface Components have no dependency to managers and any other components except the definitions inside the Business Objects. The communication between User Interface Components and other parts of the software is done through the interfaces which are implemented by User Interface Managers.

User Interface Managers: Controls of the operation of the user interface and communication with the Business Managers are provided by the User Interface Managers. By using User Interface Managers, functionality is separated from User Interface Components completely, so there are no dependencies to User Interface Components. It results in a more understandable code structure and User Interface Components are not affected from changes in business logic.

Business Managers: The business logic of the software is managed by the Business Managers. These components manage business logic and software workflow by using Business Components and other software units. For example, a Business Manager for a sensor performs operations like initiating and terminating the communication with the sensor, receiving a data from the sensor, writing received data to the database, sending data to other devices/units or displaying it on the user interface.

Business Components: These components have basic business capabilities. The calculations and algorithms related to business logic are developed as Business Components. As an example, capabilities like decoding of the data received from a sensor, internal processing of the received data and encoding of data sent to the sensor can be done by Business Components. Providing functionality by using these basic capabilities is done by Business Managers.

Business Objects: The data structure definitions that are needed by all components of the software are located under Business Objects. Event classes and the service definitions that are used in communication inside software are also located under Business Objects. When Object Relational Mapping is used for accessing the database, mapping classes to the database are also defined under Business Objects.

Abstraction Components: Abstraction Components are components that are not domain specific and capable of presenting abstractualized abilities independently of implementation details. These components provide abstraction for low-level operations that need direct interaction with the operating system, for instance:

- Printer operations
- Serial Bus Communication
- Usage of USB/CD/DVD for storage
- Accessing special ports like PIO

Application: Application layer includes classes that initialize all manager components (User Interface Managers and Business Managers) according to their starting levels and starts all operations. This layer also includes classes that contain codes related to how the software handle faults that may be encountered during startup.

Usage of these components from all layers is not free. Detailed usage rules are given under the section 3.1.1 covering the interaction in all layers.

3.1.1 Software Development Rules According to DSS Reference Software Architecture

3.1.1.1 Manager Concept

Managers are the components responsible for managing software workflow that are functionally grouped for the purposes of managing User Interface Components at the User Interface layer or managing business workflows at the Business Layer.

Managers start with the start of the application and serve as long as it is not terminated. The concept of starting and terminating can have different meanings for each manager. Creating DDS subscribers and publishers for Business Manager responsible for DDS communication or reading the configuration file and creating the relevant data structure for Business Manager responsible for the configuration operations can be given as examples of the operations performed during the initialization of the managers. During the termination, closing the opened files, terminating the network communication, etc. can be performed by Business Managers.

3.1.1.2 Data Store Concept

Data Store is an infrastructure that is developed to meet the needs of multiple managers sharing data and informing users of changes in data. Data stores can only be accessed by managers. Almost all status in the software is kept in this infrastructure.

3.1.1.3 Communication Mechanism Between Managers

The data transfer and workflow between the manager components can be performed in two different ways; asynchronous and synchronous.

Managers communicate synchronously via service interfaces with the method call. At the start of the software, services that are served by managers are injected into each manager via the relevant infrastructure component. When a manager needs to communicate with another manager, it uses the relevant service.

Managers communicate asynchronously by using event based infrastructure. Another mechanism for data transfer is data store.

3.1.1.4 Communication Mechanism Between Components

Communication mechanism between components of DSS-RSA is defined below:

- Tools can be used by all components except for Abstraction Components.
- Software Infrastructure Components can be used by all components except for Abstraction Components.
- Business Objects can be used by all components except for Abstraction Components.
- User Interface Components can not have any dependency to other components except Business Objects and User Interface Manager Interfaces.
- Access to the corresponding User Interface Manager from the User Interface Component can take place via the interface that User Interface Manager has

presented for User Interface Component.

- User Interface Manager is responsible for creating the instance of the User Interface Component and injecting the interface to the User Interface Component during the creation process.
- There is no direct access to the Abstraction Layer from User Interface Managers and User Interface Components. If there is a need for access in this direction, it is done via the relevant Business Manager.
- Business Components can only be accessed from Business Managers.
- Communication of Business Managers with external interfaces of the software or with operating system is made via a method call to the relevant Abstraction Component.
- Since Abstraction Components are developed for general purpose by considering reusability, no dependency is placed from these components to other components.

3.2 DSS Software Development Framework Generator

In addition to the source code based reusable entities, the DSS Software Team also aims to reuse other entities in the software development process. Software Development Framework Generator (DSS-Factory) contains the descriptions for how to develop software by using the reusable entities. DSS-Factory descriptions aim to accelerate the product development process and to guarantee the usage of the reusable entities.

Reusable entities used in DSS Software Team are Reference Software Architecture (DSS-RSA), Components (DSS-Component), Development Environment Infrastructures (DSS-DEI) and Knowledge Base (DSS-Knowledge Base). DSS-Factory determines how to gather these reusable entities. Developer of a software product uses the descriptions defined in DSS-Factory at the starting point. After the requirements of an application are specified, by using DSS-RSA descriptions architectural design is created. Then, developer selects the needed reusable entities and initiates development process of software framework. Figure 3.2 shows DSS-Factory process schema.

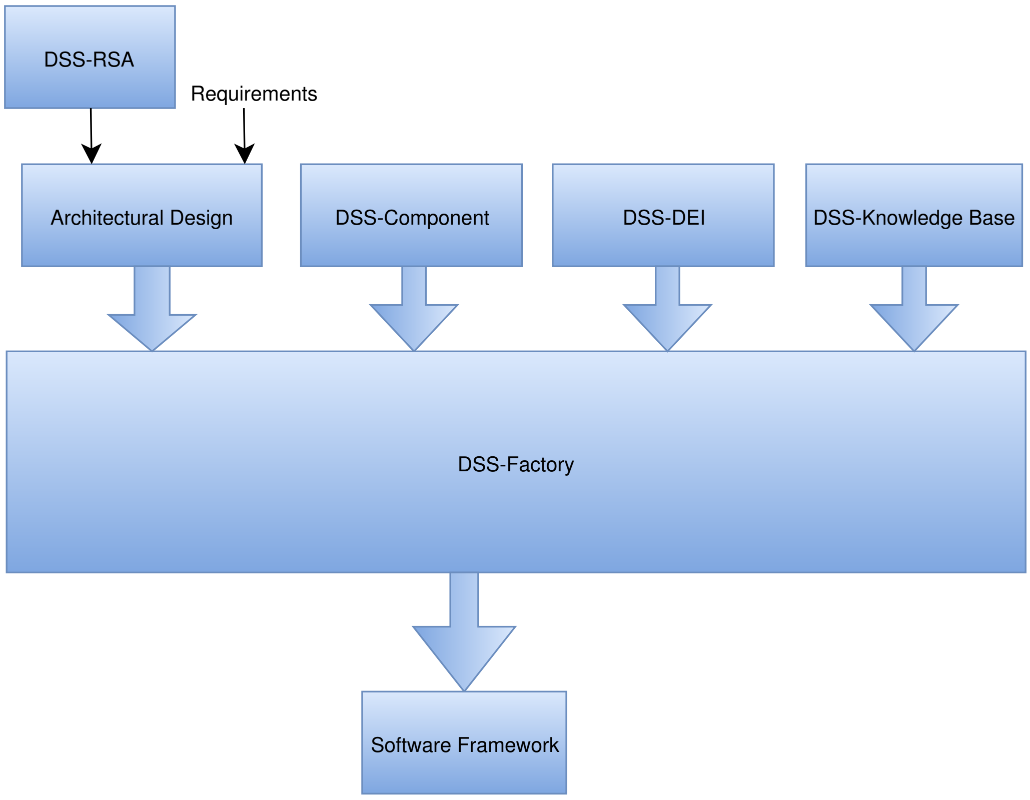


Figure 3.2: DSS-Factory process schema

Descriptions of the reusable entities in Figure 3.2 are given below:

DSS-Component: Code segments developed by DSS-Software Team, which are needed in two or more projects and do not contain any information specific to these projects, are turned into components and re-used. DSS-Components consist of common system components. DDS Abstraction Component, Operating System Abstraction Component, Serial Channel Abstraction Component, Socket Abstraction Component, External Device Access Component can be given as examples of DSS-Component.

DSS-DEI: DSS Development Environment Infrastructure corresponds to definitions used for team-wide sharing of the tools, technical standards and methods used in all phases of the software development process. Guide documents describing the use of infrastructures are prepared and template definitions are made to standardize the usage. For example, DSS-DEI defines how to integrate issues that are not defined in the SST Processes in the DSS projects for the use of relevant tools in the requirement management phase. In this way, the Requirement Management Infrastructure is used with similar approaches and more effectively in all projects. DSS-DEI also defines how to do the preparation of the integrated development environment, code configuration control, automatic version generation, code documentation, etc.

DSS-Knowledge Base: Any information produced in DSS Software Team has been accepted as reusable entity. In order to ensure the permanence of this information DSS-Knowledge Base has been established. DSS-Knowledge Base corresponds to the ready-made infrastructures for Software Engineering and Naval Defense Systems application areas or to the infrastructures created for the storage and subsequent access of the information produced as a result of the research conducted. Documents created by the DSS Software team as well as useful information that is readily available are also stored in DSS-Knowledge Base. Training notes, articles, presentations, thesis studies, research reports and standards can be given as examples of information stored in DSS-Knowledge Base.

In the scope of this study, only the first phase of DSS-Factory is discussed, which is automatic generation of skeleton codes after software architectural design is created in compliance with DSS-RSA.

CHAPTER 4

METAMODEL FOR DSS REFERENCE SOFTWARE ARCHITECTURE

Reference Software Architecture is a special type of software architecture that can be used to instantiate concrete software architectures. Reference Software Architecture addresses a specific application domain whereas a concrete architecture is for a particular product, application or system within that domain. Naval command and control systems is a domain; a weapon-target assignment system of a warship is an example system belonging to that domain.

According to the conducted studies in literature, there is a necessity for software tools to support the Reference Software Architecture representation. Like Reference Software Architectures, metamodels in Model Driven Engineering are used to represent concrete models. There are a lot of tools and representation tecniques available for Model Driven Engineering. Reference Software Architectures and metamodels are similar concepts that both of which are abstract solutions for a family of concrete models and concrete models can be derived from both of them. Thus, available tools and representation techniques for metamodels can be used to support Reference Software Architecture representation [7].

In this chapter, metamodeling concept, metamodeling language concept, DSS-RSA Metamodel and mapping from DSS-RSA to Metamodel are described. Firstly, in Section 4.1, metamodeling concept and metamodeling levels are presented. In Section 4.2 metamodeling language concept and Eclipse EMF metamodeling language is given. In Section 4.3, DSS-RSA Metamodel, its elements and relationships are given. Lastly, in section 4.4 mapping from DSS-RSA to metamodel is described in detail.

4.1 Metamodeling Concept

In Model Driven Engineering, models play an important role and they are represented as instances of some more abstract models. In the same way, models are abstraction of real world entities. Figure 4.1 shows the levels of metamodeling. By using meta-metamodel at M3 level, metamodel at M2 level can be defined. By using metamodel at M2 level, models at M1 level can be instantiated [2].

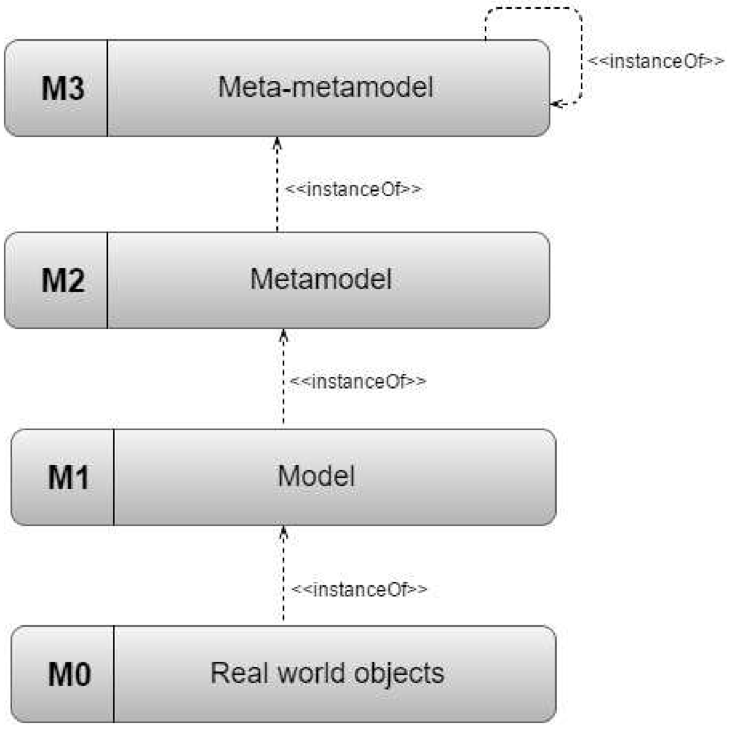


Figure 4.1: Metamodeling levels [2]

Metamodels only define abstract syntaxes of the modeling languages that they represent. Abstract syntax describes modeling concepts (classes, attributes and associations) and their properties. Metamodels do not define the concrete syntax. Concrete syntax is a notation in which graphical or textual elements are used to present the model elements. To specify the concrete syntax, additional artifacts which refer to the model elements are used. It is possible to define both graphical and textual concrete syntax for the same modeling language. Metamodels partially describe modeling constraints. For example, cardinality constraints, association ends and types for attributes can be defined by metamodel but more complicated constraints can not be expressed. For example uniqueness of name of a model element can only be expressed by a constraint language [2].

4.2 Metamodeling Language Concept

Meta Object Facility (MOF) is a standard for metamodeling language which is defined by Object Management Group (OMG). As well as MOF, there are various languages and tools that support metamodeling and other modeling capabilities like transformations (for example model-to-model or model-to-text), integration with the software development process. Eclipse development environment is one of the prominent tooling platform in Model Driven Engineering (MDE) world. It comprises popular components and tools for all modeling tasks. For MDD, the Eclipse Modeling Framework (EMF) is the core technology in Eclipse development environment. EMF is an open-source technology and good representative of MDD tools for reasons described below [2].

- EMF uses Ecore metamodeling language to define metamodels
- EMF has code generation capabilities from metamodels
- EMF has Java-based API for manipulating models
- EMF provides tree-based modeling editors to build models.
- EMF has API to serialize and deserialize models to/from XMI. By this way exchanges can be possible between tools supporting the same meta-metamodel.

In Figure 4.1, meta-metamodels at M3 level defines metamodeling languages that specify metamodeling concepts used to define metamodels at M2 level. At M2 level, metamodels represent modeling languages that specify modeling concepts used to define models. While creating a metamodel, which defines abstract syntax of domain specific modeling language, a metamodeling language is used.

4.3 DSS-RSA Metamodel

In this study, we create the metamodel of DSS Reference Software Architecture. To create DSS-RSA metamodel, Ecore metamodeling language is used. DSS-RSA metamodeling process is started of with some elements of DSS-RSA and relationships between these elements. As the first step, modeling concepts are determined.

Table 4.1 shows the modeling concept table that is transformable into an Ecore metamodel.

Table 4.1: Modeling concept table for DSS-RSA

Concept	Intrinsic Properties	Extrinsic Properties
DSSML : EClass	anaPaket : EString	Arbitrary number of IsYonetici Arbitrary number of KaYonetici Arbitrary number of KaBileseni Arbitrary number of ArayuzKay Arbitrary number of IsBileseni Arbitrary number of Veri Arbitrary number of VeriDeposu Arbitrary number of Olay Arbitrary number of Servis
KaYonetici : EClass	isim : EString seviye : EInteger paket : EString aciklama : EString	One or more of KaBileseni defined as ka One or more of ArayuzKay defined as arayuz Arbitrary number of Olay defined as yakalananOlay Arbitrary number of Servis defined as sunulanServis Arbitrary number of Servis defined as kullanilanServis Arbitrary number of VeriDeposu defined as aboneVd

Table 4.1 Continued

Concept	**Intrinsic Properties**	**Extrinsic Properties**
KaBileseni : EClass	isim : EString paket : EString aciklama : EString	One ArayuzKay defined as yoneticiArayuz
ArayuzKay : EClass	isim : EString paket : EString aciklama : EString	
IsBileseni : EClass	isim : EString paket : EString aciklama : EString	
Veri : EClass	isim : EString paket : EString tip : VeriTipi aciklama : String	
VeriDeposu : EClass	isim : EString tip : VeriDeposuTipi aciklama : EString	One Veri as veri1 Zero or One Veri as veri2
IsYonetici : EClass	isim : EString seviye : EInteger paket : EString aciklama : EString	Arbitrary number of Olay defined as yakalananOlay Arbitrary number of Servis defined as sunulanServis Arbitrary number of Servis defined as kullanilanServis Arbitrary number of VeriDeposu defined as aboneVd
Servis : EClass	isim : EString tip : ServisTipi aciklama : EString	
Olay: EClass	isim : EString aciklama : EString	
VeriDeposuTipi :EEnum	Yapi : EEnumLiteral Dizi : EEnumLiteral	

Table 4.1 Continued

Concept	**Intrinsic Properties**	**Extrinsic Properties**
VeriTipi :EEnum	Class : EEnumLiteral Enum : EEnumLiteral Interface : EEnumLiteral	
ServisTipi :EEnum	Isy : EEnumLiteral Kay : EEnumLiteral	

Concepts in Table 4.1 are transformed into classes (EClass in Ecore Metamodel) or enumerations (EEnum in Ecore metamodel). Intrinsic properties are transformed into attributes. For attributes, types have to be presented, such as String as EString in Ecore metamodel or Integer as EInteger in Ecore metamodel. If there is a range of possible values, enumerations are defined, such as $VeriDeposuTipi$, $VeriTipi$ and $ServisTipi$ in Table 4.1. Attributes of enumerations are defined as EEnumLiteral in Ecore metamodel. Extrinsic properties are transformed into associations between classes. For the association ends, upper and lower bounds of multiplicities have to be set properly [2].

Figure 4.2 shows the metamodel of DSS-RSA. Development steps of metamodel as EMF Ecore Model are given in Appendix A in detail.

4.4 Mapping from DSS-RSA to Metamodel

- User Interface Layer of DSS-RSA includes User Interface Components and User Interface Managers. In the metamodel KaBileseni elements are defined to express User Interface Components, KaYonetici elements are defined to express User Interface Managers. Management of User Interface functionality is handled by User Interface Managers in DSS-RSA, in the metamodel management of KaBileseni elements is done by KaYonetici elements.

- The communication between User Interface Components and other parts of the software is done through the interfaces which are implemented by User Interface Managers in DSS-RSA. Communication between KaBileseni and KaYonetici elements is provided by the ArayuzKay elements in the metamodel.

- Business Layer of DSS-RSA includes Business Managers and Business Components. In the metamodel, IsYonetici and IsBileseni elements are defined for Business Layer. IsYonetici elements represent Business Managers and IsBileseni elements represent Business Components.

- Business Objects in DSS-RSA include data structure definitions which are defined as Veri, events which are defined as Olay and services which are defined as Servis in the metamodel.

- Data stores in DSS-RSA are represented as VeriDeposu in the metamodel.

- In DSS-RSA, Business Managers implement services, which reside in Business Objetcs, to allow other managers of the software to communicate with themselves. To express this functionality in the metamodel, we create $sunulanServis$ relationship between IsYonetici and Servis elements.

- User Interface Managers implement services, which reside in Business Objetcs, to allow other managers of the software to communicate with themselves. To express this functionality in the metamodel, we create $sunulanServis$ relationship between KaYonetici and Servis elements.

- To communicate with Business Managers or User Interface Managers, other manager elements use services which reside in Business Objects in DSS-RSA. To express this functionality, we create relationships, which are called as $kullanilanServis$, between IsYonetici and Servis elements and between KaYonetici and Servis elements in the metamodel.

- Manager elements can communicate asynchronously with other manager elements by using event based infrastructure in DSS-RSA. To provide asynchronous communication between managers, we create relationships, which are called as $yakalananOlay$, between IsYonetici and Olay elements and between KaYonetici and Olay elements in the metamodel.

- Another asynchronized communication mechanism between manager elements in DSS-RSA is data store. To express this communication, we create relationships, which are called as $aboneVd$, between IsYonetici and VeriDeposu elements and between KaYonetici and VeriDeposu elements in the metamodel.

- In DSS-RSA, data stores can be defined as structure or sequence by using data structure definitions in Business Objects. In the metamodel, VeriDeposu elements have an attribute called Tip. Tip attribute values can be $Yapi$ or $Dizi$ in the metamodel. Regardless of whether the value is $Yapi$ or $Dizi$, VeriDeposu element must have a relationship called as $veri1$ with Veri element. If Tip attribute value is $Dizi$, VeriDeposu element must also have a relationship called as $veri2$ with Veri element. If Tip attribute value is $Yapi$, $veri2$ relationship can not be established between VeriDeposu and Veri element.

- If application will have a user interface, defined User Interface Managers must have at least one User Interface Component and at least one Interface. For this reason, we put numerical restrictions in the metamodel. Each KaYonetici element must have a relationship with at least one ArayuzKay called as $arayuz$ and KaBileseni element called as ka. Each KaBileseni element must have a relationship with one ArayuzKay element called as $yoneticiArayuz$.

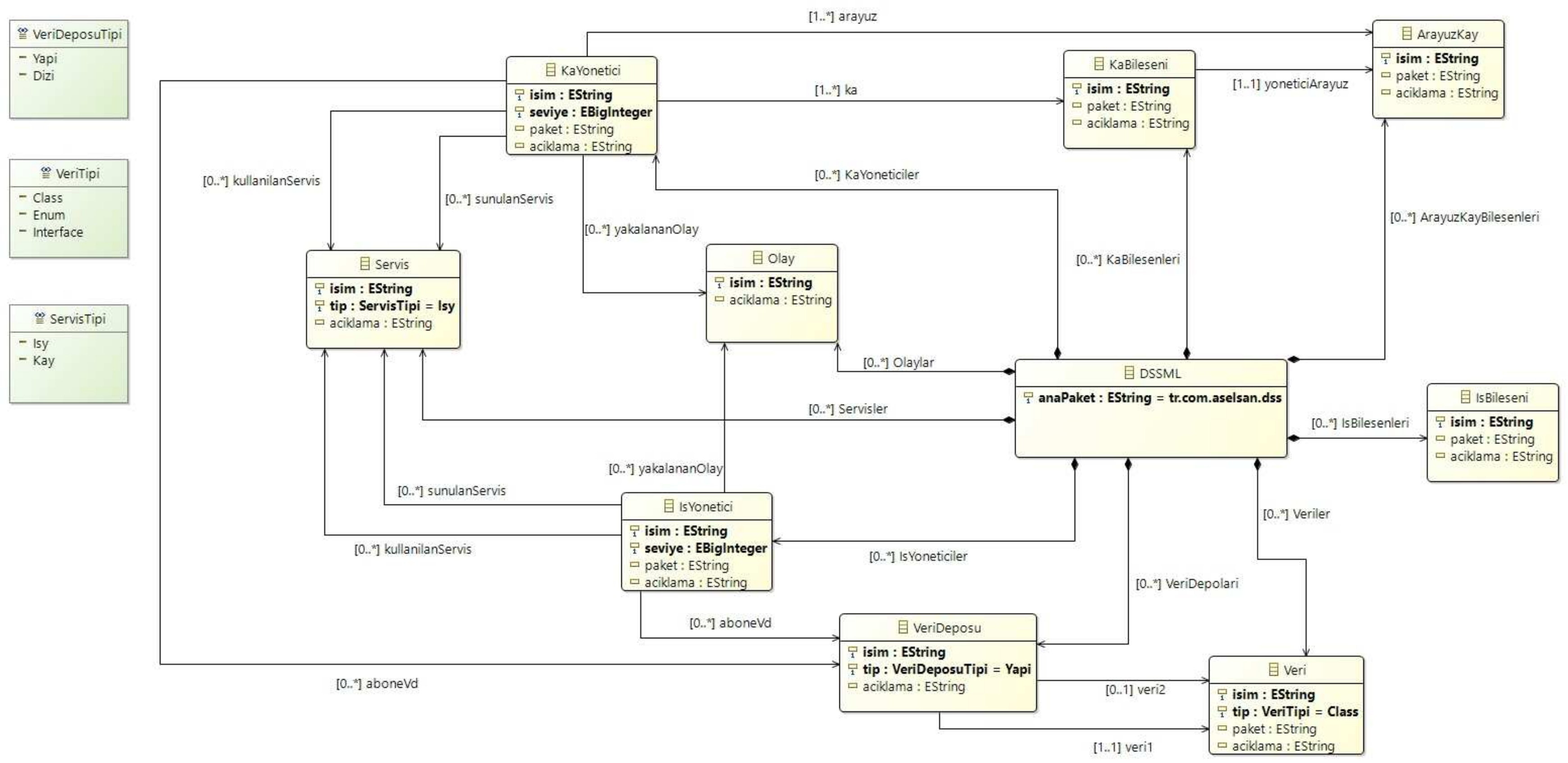


Figure 4.2: DSS-RSA Metamodel

CHAPTER 5

STATIC SEMANTIC RULE DEFINITION WITH OCL

Metamodeling languages can only express part of the relevant information required to define a modeling language. These very basic modeling constraints expressed by metamodeling languages are cardinality constraints. Cardinality constraints restrict relationships between elements in metamodels [2]. For example in DSS-RSA Metamodel, each KaYonetici element can have ka relationship with at least one KaBileseni element and $arayuz$ relationship with at least one ArayuzKay element.

There is a need to describe additional constraints about the elements in the model. Static semantic rules define well-formedness of a metamodel. These well-formedness rules are used for both defining constraints on how models can be formed, and validating the models constructed upon a specific metamodel [13]. For restrictions that cannot be brought in metamodeling language, some static semantic rules have been identified by using Object Constraint Language (OCL).

OCL is a general purpose formal language which is easy to read and write. It has been developed as a business modeling language within the IBM Insurance division. It is adopted as a standard by OMG. OCL is a typed and side effect free specification language. When an OCL expression is evaluated, it simply returns a value, does not change the state of the system. Each OCL expression has a type. An OCL expression must conform to the type conformance rules and operations of that type [19].

In Eclipse, to define OCL constraints for Ecore based metamodels several plugins are available. Eclipse OCL is an implementation of the OMG OCL 2.4 specification for use with Ecore metamodels. It provides APIs for parsing and evaluating OCL constraints and queries on Ecore Models [20].

Eclipse OCL supports OCL expressions embedded within Ecore using the OCLinEcore editor. The OCLinEcore editor has been available for use since Eclipse OCL 3.0.0 by installing the Eclipse OCL Examples and Editors functionality [21].

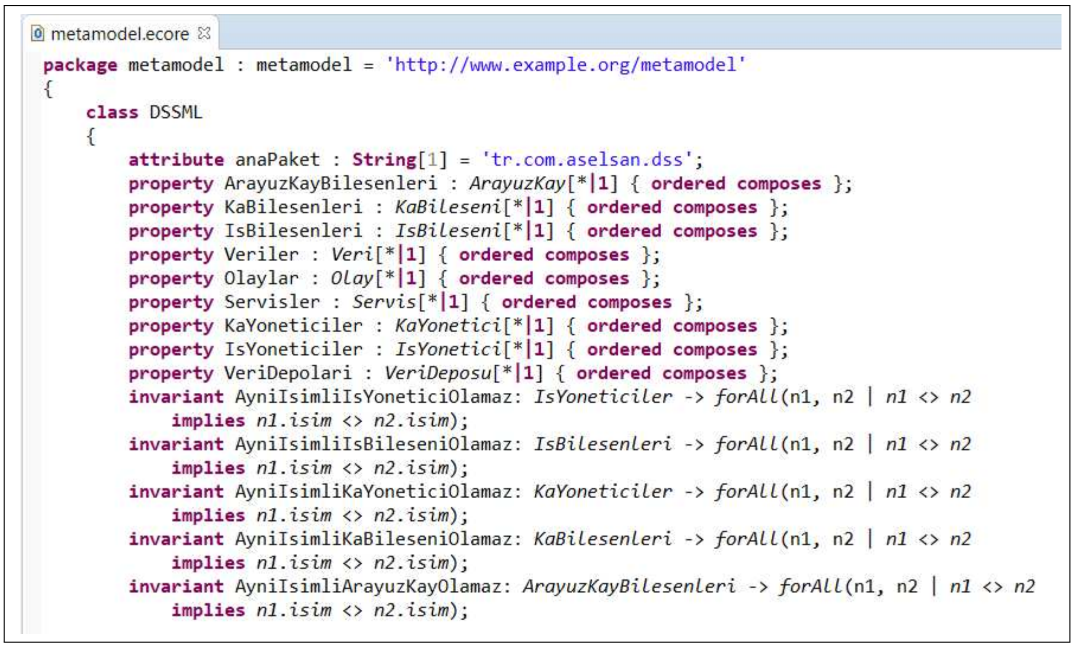
In this chapter, OCLinEcore Editor and defined OCL constraints are described. Firstly, in Section 5.1, OCLinEcore Editor is introduced. In Section 5.2 defined naming constraints, in Section 5.3 defined cardinality constraints and in Section 5.4 defined element usage constraints are expressed. In Section 5.5 defined constraints to check the usage of Servis elements by KaYonetici and IsYonetici elements, in Section 5.6 defined IsYonetici element constraints, in Section 5.7 defined IsBileseni element constraints, in Section 5.8 defined KaYonetici element constraints, in Section 5.9 defined KaBileseni element constraints, in Section 5.10 defined ArayuzKay element constraints, in Section 5.11 defined Olay element constraints, in Section 5.12 defined Servis element constraints and lastly, in Section 5.13 contraints defined for VeriDeposu elements in OCL are described in detail.

5.1 OCLinEcore Editor

We define some OCL constraints by using Eclipse OCL project. After Ecore metamodel is defined, we open metamodel with OCLinEcore editor. Figure 5.1 shows a part of DSS-RSA metamodel opened in OCLinEcore editor.

Constraints in OCL are represented as Invariant. The name of the constraint is written next to this keyword. To access properties of an element or related elements of an element in metamodel, dot notation is used. The standard OCL Library predefines the primitive and collection types and operations, quantifiers (like forAll, exists, etc.) and iterators (like select, etc.) [2].

OCL constraints are checked after application architecture model is created by using created graphical modeling editor as described in Chapter 6. To validate the model, design area is right clicked and "$OCL \rightarrow Validate$" is selected. If some OCL constraints are violated, validation results show problems. Chapter 8 describes how we check defined OCL constraints for the conducted case study.

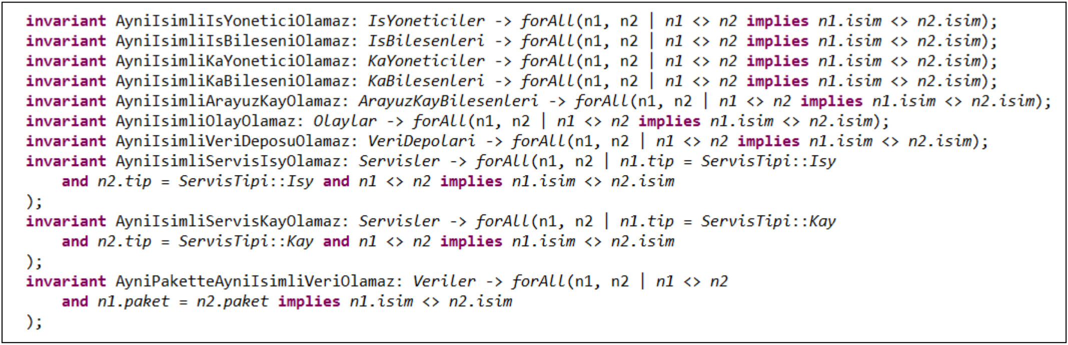


```
metamodel.ecore
package metamodel : metamodel = 'http://www.example.org/metamodel'
{
    class DSSML
    {
        attribute anaPaket : String[1] = 'tr.com.aselsan.dss';
        property ArayuzKayBilesenleri : ArayuzKay[*|1] { ordered composes };
        property KaBilesenleri : KaBileseni[*|1] { ordered composes };
        property IsBilesenleri : IsBileseni[*|1] { ordered composes };
        property Veriler : Veri[*|1] { ordered composes };
        property Olaylar : Olay[*|1] { ordered composes };
        property Servisler : Servis[*|1] { ordered composes };
        property KaYoneticiler : KaYonetici[*|1] { ordered composes };
        property IsYoneticiler : IsYonetici[*|1] { ordered composes };
        property VeriDepolari : VeriDeposu[*|1] { ordered composes };
        invariant AyniIsimliIsYoneticiOlamaz: IsYoneticiler -> forAll(n1, n2 | n1 <> n2
            implies n1.isim <> n2.isim);
        invariant AyniIsimliIsBileseniOlamaz: IsBilesenleri -> forAll(n1, n2 | n1 <> n2
            implies n1.isim <> n2.isim);
        invariant AyniIsimliKaYoneticiOlamaz: KaYoneticiler -> forAll(n1, n2 | n1 <> n2
            implies n1.isim <> n2.isim);
        invariant AyniIsimliKaBileseniOlamaz: KaBilesenleri -> forAll(n1, n2 | n1 <> n2
            implies n1.isim <> n2.isim);
        invariant AyniIsimliArayuzKayOlamaz: ArayuzKayBilesenleri -> forAll(n1, n2 | n1 <> n2
            implies n1.isim <> n2.isim);
```

Figure 5.1: Some OCL constraints defined in OCLinEcore editor

5.2 OCL Naming Constraints

We define some naming constraints to our DSS-RSA metamodel shown in Figure 5.2.



```
invariant AyniIsimliIsYoneticiOlamaz: IsYoneticiler -> forAll(n1, n2 | n1 <> n2 implies n1.isim <> n2.isim);
invariant AyniIsimliIsBileseniOlamaz: IsBilesenleri -> forAll(n1, n2 | n1 <> n2 implies n1.isim <> n2.isim);
invariant AyniIsimliKaYoneticiOlamaz: KaYoneticiler -> forAll(n1, n2 | n1 <> n2 implies n1.isim <> n2.isim);
invariant AyniIsimliKaBileseniOlamaz: KaBilesenleri -> forAll(n1, n2 | n1 <> n2 implies n1.isim <> n2.isim);
invariant AyniIsimliArayuzKayOlamaz: ArayuzKayBilesenleri -> forAll(n1, n2 | n1 <> n2 implies n1.isim <> n2.isim);
invariant AyniIsimliOlayOlamaz: Olaylar -> forAll(n1, n2 | n1 <> n2 implies n1.isim <> n2.isim);
invariant AyniIsimliVeriDeposuOlamaz: VeriDepolari -> forAll(n1, n2 | n1 <> n2 implies n1.isim <> n2.isim);
invariant AyniIsimliServisIsyOlamaz: Servisler -> forAll(n1, n2 | n1.tip = ServisTipi::Isy
    and n2.tip = ServisTipi::Isy and n1 <> n2 implies n1.isim <> n2.isim
);
invariant AyniIsimliServisKayOlamaz: Servisler -> forAll(n1, n2 | n1.tip = ServisTipi::Kay
    and n2.tip = ServisTipi::Kay and n1 <> n2 implies n1.isim <> n2.isim
);
invariant AyniPaketteAyniIsimliVeriOlamaz: Veriler -> forAll(n1, n2 | n1 <> n2
    and n1.paket = n2.paket implies n1.isim <> n2.isim
);
```

Figure 5.2: OCL naming constraints

Naming constraints shown in Figure 5.2 are explained in detail below:

- For each IsYonetici element pairs, $isim$ attribute values must be different from each other. This rule is expressed in OCL with "AyniIsimliIsYoneticiOlamaz" invariant.

- For each IsBileseni element pairs, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniIsimliIsBileseniOlamaz” invariant.

- For each KaYonetici element pairs, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniIsimliKaYoneticiOlamaz” invariant.

- For each KaBileseni element pairs, $isim$ attribute values must be different from each other. This rule is represented in OCL with “AyniIsimliKaBileseniOlamaz” invariant.

- For each ArayuzKay element pairs, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniIsimliArayuzKayOlamaz” invariant.

- For each Olay element pairs, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniIsimliOlayOlamaz” invariant.

- For each VeriDeposu element pairs, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniIsimliVeriDeposuOlamaz” invariant.

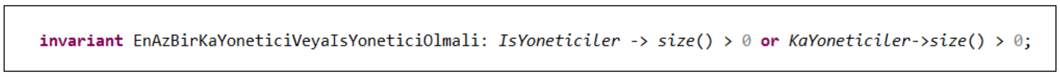
- For each Servis element pairs having the tip attribute value of Isy, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniIsimliServisIsyOlamaz” invariant.

- For each Servis element pairs having the tip attribute value of Kay, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniIsimliServisKayOlamaz” invariant.

- For each Veri element pairs having the same $paket$ attribute values, $isim$ attribute values must be different from each other. This rule is expressed in OCL with “AyniPaketteAyniIsimliVeriOlamaz” invariant.

5.3 OCL Cardinality Constraints

We define a cardinality constraint that could not be defined in metamodel shown in Figure 5.3.



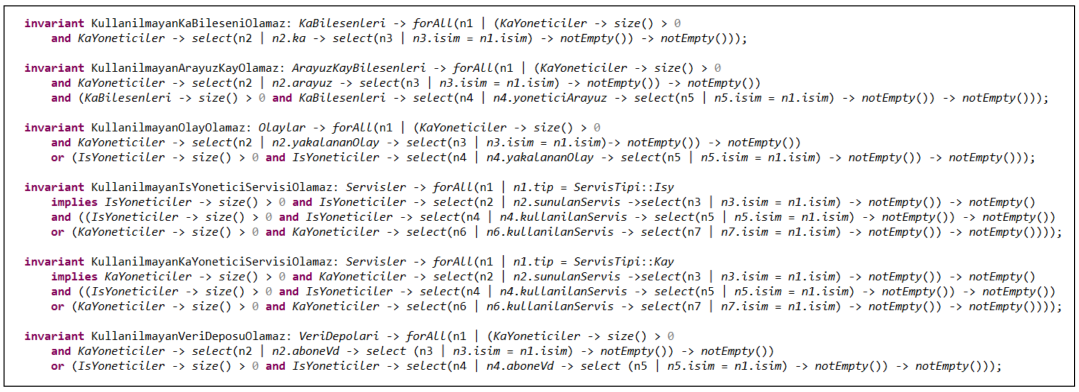
```
invariant EnAzBirKaYoneticiVeyaIsYoneticiOlmali: IsYoneticiler -> size() > 0 or KaYoneticiler->size() > 0;
```

Figure 5.3: OCL cardinality constraint

In a DSS-RSA model, there must be at least one KaYonetici or at least one IsYonetici element. This rule is expressed in OCL with “EnAzBirKaYoneticiVeyaIsYoneticiOlmali” invariant.

5.4 OCL Element Usage Constraints

We define some constraints to check the usage of elements in DSS-RSA metamodel shown in Figure 5.4.



```
invariant KullanilmayanKaBileseniOlamaz: KaBilesenleri -> forAll(n1 | (KaYoneticiler -> size() > 0
    and KaYoneticiler -> select(n2 | n2.ka -> select(n3 | n3.isim = n1.isim) -> notEmpty()) -> notEmpty()));

invariant KullanilmayanArayuzKayOlamaz: ArayuzKayBilesenleri -> forAll(n1 | (KaYoneticiler -> size() > 0
    and KaYoneticiler -> select(n2 | n2.arayuz -> select(n3 | n3.isim = n1.isim) -> notEmpty()) -> notEmpty())
    and (KaBilesenleri -> size() > 0 and KaBilesenleri -> select(n4 | n4.yoneticiArayuz -> select(n5 | n5.isim = n1.isim) -> notEmpty()) -> notEmpty()));

invariant KullanilmayanOlayOlamaz: Olaylar -> forAll(n1 | (KaYoneticiler -> size() > 0
    and KaYoneticiler -> select(n2 | n2.yakalananOlay -> select(n3 | n3.isim = n1.isim)-> notEmpty()) -> notEmpty())
    or (IsYoneticiler -> size() > 0 and IsYoneticiler -> select(n4 | n4.yakalananOlay -> select(n5 | n5.isim = n1.isim) -> notEmpty()) -> notEmpty()));

invariant KullanilmayanIsYoneticiServisiOlamaz: Servisler -> forAll(n1 | n1.tip = ServisTipi::Isy
    implies IsYoneticiler -> size() > 0 and IsYoneticiler -> select(n2 | n2.sunulanServis ->select(n3 | n3.isim = n1.isim) -> notEmpty()) -> notEmpty()
    and ((IsYoneticiler -> size() > 0 and IsYoneticiler -> select(n4 | n4.kullanilanServis -> select(n5 | n5.isim = n1.isim) -> notEmpty()) -> notEmpty())
    or (KaYoneticiler -> size() > 0 and KaYoneticiler -> select(n6 | n6.kullanilanServis -> select(n7 | n7.isim = n1.isim) -> notEmpty()) -> notEmpty())));

invariant KullanilmayanKaYoneticiServisiOlamaz: Servisler -> forAll(n1 | n1.tip = ServisTipi::Kay
    implies KaYoneticiler -> size() > 0 and KaYoneticiler -> select(n2 | n2.sunulanServis ->select(n3 | n3.isim = n1.isim) -> notEmpty()) -> notEmpty()
    and ((IsYoneticiler -> size() > 0 and IsYoneticiler -> select(n4 | n4.kullanilanServis -> select(n5 | n5.isim = n1.isim) -> notEmpty()) -> notEmpty())
    or (KaYoneticiler -> size() > 0 and KaYoneticiler -> select(n6 | n6.kullanilanServis -> select(n7 | n7.isim = n1.isim) -> notEmpty()) -> notEmpty())));

invariant KullanilmayanVeriDeposuOlamaz: VeriDepolari -> forAll(n1 | (KaYoneticiler -> size() > 0
    and KaYoneticiler -> select(n2 | n2.aboneVd -> select (n3 | n3.isim = n1.isim) -> notEmpty()) -> notEmpty())
    or (IsYoneticiler -> size() > 0 and IsYoneticiler -> select(n4 | n4.aboneVd -> select (n5 | n5.isim = n1.isim) -> notEmpty()) -> notEmpty()));
```

Figure 5.4: OCL element usage constraints

Usage constraints shown in Figure 5.4 are explained in detail below:

- For each KaBileseni element defined in a DSS-RSA model, there must be a KaYonetici element which has a ka relationship with this KaBileseni element.

This rule is expressed in OCL with "KullanilmayanKaBileseniOlamaz" invariant.

- For each ArayuzKay element defined in a DSS-RSA model, there must be a KaYonetici element which has an $arayuz$ relationship with this ArayuzKay element and there must be a KaBileseni element which has a $yoneticiArayuz$ relationship with this ArayuzKay element. This rule is expressed in OCL with "KullanilmayanArayuzKayOlamaz" invariant.

- For each Olay element defined in a DSS-RSA model, there must be at least one KaYonetici or IsYonetici element which has a $yakalananOlay$ relationship with this Olay element. This rule is expressed in OCL with "KullanilmayanOlayOlamaz" invariant.

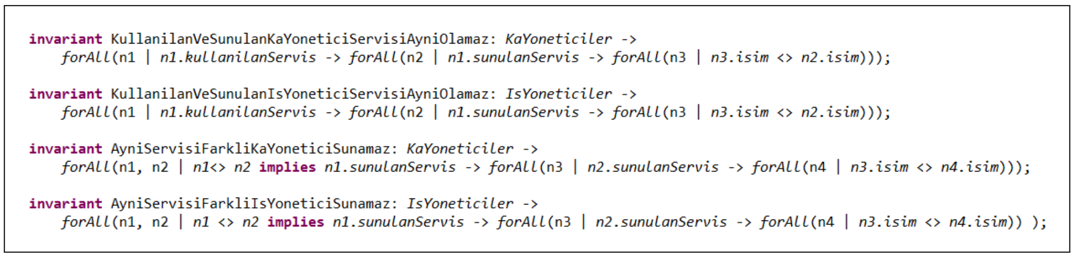
- For each Servis element defined in a DSS-RSA model, if tip attribute has the value of Isy, there must be an IsYonetici element which has a $sunulanServis$ relationship with this Servis element and there must be at least one IsYonetici or KaYonetici element which has a $kullanilanServis$ relationship with this Servis element. This rule is expressed in OCL with "KullanilmayanIsYoneticiServisiOlamaz" invariant.

- For each Servis element defined in a DSS-RSA model, if tip attribute has the value of Kay, there must be a KaYonetici element which has a $sunulanServis$ relationship with this Servis element and there must be at least one KaYonetici or IsYonetici element which has a $kullanilanServis$ relationship with this Servis element. This rule is expressed in OCL with "KullanilmayanKaYoneticiServisiOlamaz" invariant.

- For each VeriDeposu element defined in a DSS-RSA model, there must be at least one KaYonetici or IsYonetici element which has an $aboneVd$ relationship with this VeriDeposu element. This rule is expressed in OCL with "KullanilmayanVeriDeposuOlamaz" invariant.

5.5 OCL Servis Element Usage Constraints

We define some constraints for usage of Servis Elements by KaYonetici and IsYonetici elements in a DSS-RSA model. These constraints are shown in Figure 5.5.



```
invariant KullanilanVeSunulanKaYoneticiServisiAyniOlamaz: KaYoneticiler ->
    forAll(n1 | n1.kullanilanServis -> forAll(n2 | n1.sunulanServis -> forAll(n3 | n3.isim <> n2.isim)));

invariant KullanilanVeSunulanIsYoneticiServisiAyniOlamaz: IsYoneticiler ->
    forAll(n1 | n1.kullanilanServis -> forAll(n2 | n1.sunulanServis -> forAll(n3 | n3.isim <> n2.isim)));

invariant AyniServisiFarkliKaYoneticiSunamaz: KaYoneticiler ->
    forAll(n1, n2 | n1<> n2 implies n1.sunulanServis -> forAll(n3 | n2.sunulanServis -> forAll(n4 | n3.isim <> n4.isim)));

invariant AyniServisiFarkliIsYoneticiSunamaz: IsYoneticiler ->
    forAll(n1, n2 | n1 <> n2 implies n1.sunulanServis -> forAll(n3 | n2.sunulanServis -> forAll(n4 | n3.isim <> n4.isim)) );
```

Figure 5.5: OCL Servis element usage constraints

Servis element usage constraints shown in Figure 5.5 are explained in detail below:

- For each KaYonetici element defined in a DSS-RSA model, Servis elements that KaYonetici element has $kullanilanServis$ relationship with and $sunulanServis$ relationship with must be different from each other. This rule is expressed in OCL with "KullanilanVeSunulanKaYoneticiServisiAyniOlamaz" invariant.

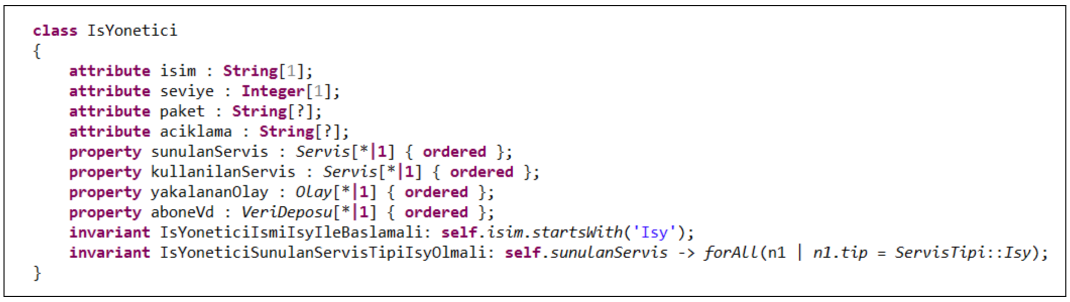
- For each IsYonetici element defined in a DSS-RSA model, Servis elements that IsYonetici element has $kullanilanServis$ relationship with and $sunulanServis$ relationship with must be different from each other. This rule is expressed in OCL with "KullanilanVeSunulanIsYoneticiServisiAyniOlamaz" invariant.

- For each KaYonetici element pairs defined in a DSS-RSA model, Servis elements that KaYonetici element pairs have $sunulanServis$ relationship with must be different from each other. This rule is expressed in OCL with "AyniServisiFarkliKaYoneticiSunamaz" invariant.

- For each IsYonetici element pairs defined in a DSS-RSA model, Servis elements that IsYonetici element pairs have $sunulanServis$ relationship with must be different from each other. This rule is expressed in OCL with "AyniServisiFarkliIsYoneticiSunamaz" invariant.

5.6 OCL IsYonetici Element Constraints

We define some constraints for IsYonetici elements in a DSS-RSA model. These constraints are shown in Figure 5.6.



```
class IsYonetici
{
    attribute isim : String[1];
    attribute seviye : Integer[1];
    attribute paket : String[?];
    attribute aciklama : String[?];
    property sunulanServis : Servis[*|1] { ordered };
    property kullanilanServis : Servis[*|1] { ordered };
    property yakalananOlay : Olay[*|1] { ordered };
    property aboneVd : VeriDeposu[*|1] { ordered };
    invariant IsYoneticiIsmiIsyIleBaslamali: self.isim.startsWith('Isy');
    invariant IsYoneticiSunulanServisTipiIsyOlmali: self.sunulanServis -> forAll(n1 | n1.tip = ServisTipi::Isy);
}
```

Figure 5.6: OCL IsYonetici element constraints

IsYonetici element constraints shown in Figure 5.6 are explained in detail below:

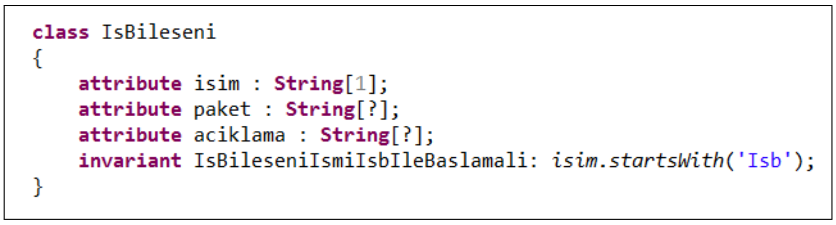
- $isim$ attribute values of each IsYonetici element must start with Isy. This rule is expressed in OCL with "IsYoneticiIsmiIsyIleBaslamali" invariant.

- For each IsYonetici element defined in a DSS-RSA model, Servis elements that IsYonetici element has $sunulanServis$ relationship with must have tip attribute value of Isy. This rule is expressed in OCL with "IsYoneticiSunulanServisTipiIsyOlmali" invariant.

5.7 OCL IsBileseni Element Constraints

We define a constraint for IsBileseni elements in a DSS-RSA model. This constraint is shown in Figure 5.7.

IsBileseni element constraint shown in Figure 5.7 is that $isim$ attribute value of each IsBileseni element must start with Isb. This rule is expressed in OCL with "IsBileseniIsmiIsbIleBaslamali" invariant.

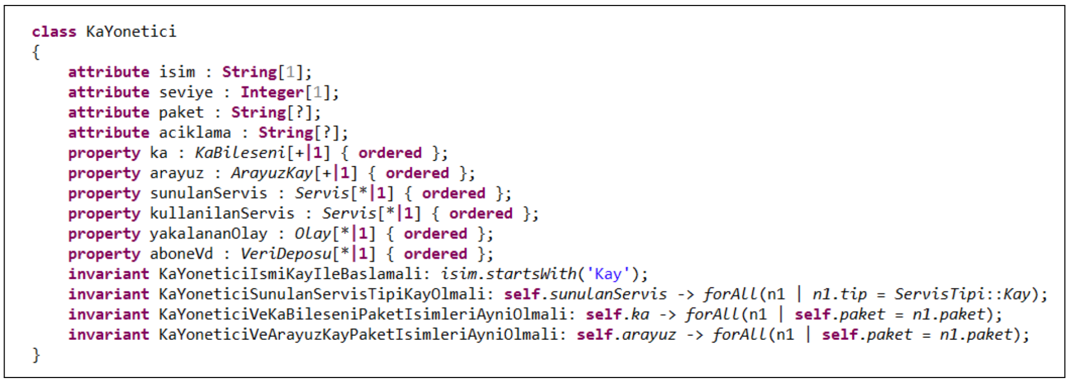


```
class IsBileseni
{
    attribute isim : String[1];
    attribute paket : String[?];
    attribute aciklama : String[?];
    invariant IsBileseniIsmiIsbIleBaslamali: isim.startsWith('Isb');
}
```

Figure 5.7: OCL IsBileseni element constraint

5.8 OCL KaYonetici Element Constraints

We define some constraints for KaYonetici elements in a DSS-RSA model. These constraints are shown in Figure 5.8.



```
class KaYonetici
{
    attribute isim : String[1];
    attribute seviye : Integer[1];
    attribute paket : String[?];
    attribute aciklama : String[?];
    property ka : KaBileseni[+|1] { ordered };
    property arayuz : ArayuzKay[+|1] { ordered };
    property sunulanServis : Servis[*|1] { ordered };
    property kullanilanServis : Servis[*|1] { ordered };
    property yakalananOlay : Olay[*|1] { ordered };
    property aboneVd : VeriDeposu[*|1] { ordered };
    invariant KaYoneticiIsmiKayIleBaslamali: isim.startsWith('Kay');
    invariant KaYoneticiSunulanServisTipiKayOlmali: self.sunulanServis -> forAll(n1 | n1.tip = ServisTipi::Kay);
    invariant KaYoneticiVeKaBileseniPaketIsimleriAyniOlmali: self.ka -> forAll(n1 | self.paket = n1.paket);
    invariant KaYoneticiVeArayuzKayPaketIsimleriAyniOlmali: self.arayuz -> forAll(n1 | self.paket = n1.paket);
}
```

Figure 5.8: OCL KaYonetici element constraints

KaYonetici element constraints shown in Figure 5.8 are explained in detail below:

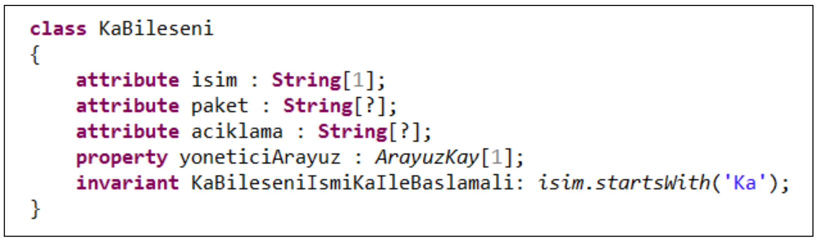
- $isim$ attribute value of each KaYonetici element must start with Kay. This rule is expressed in OCL with "KaYoneticiIsmiKayIleBaslamali" invariant.
- For each KaYonetici element defined in a DSS-RSA model, Servis elements that KaYonetici element has $sunulanServis$ relationship with must have tip attribute value of Kay. This rule is expressed in OCL with "$KaYoneticiSunulanServisTipiKayOlmali$" invariant.
- For each KaYonetici element defined in a DSS-RSA model, KaBileseni elements that KaYonetici element has ka relationship with must have the same $paket$ attribute value with KaYonetici element. This rule is expressed in OCL

with “KaYoneticiVeKaBileseniPaketIsimleriAyniOlmali” invariant.

- For each KaYonetici element defined in a DSS-RSA model, ArayuzKay elements that KaYonetici element has $arayuz$ relationship with must have the same $paket$ attribute value with KaYonetici element. This rule is expressed in OCL with “KaYoneticiVeArayuzKayPaketIsimleriAyniOlmali” invariant.

5.9 OCL KaBileseni Element Constraints

We define a constraint for KaBileseni elements in a DSS-RSA model. This constraint is shown in Figure 5.9.



```
class KaBileseni
{
    attribute isim : String[1];
    attribute paket : String[?];
    attribute aciklama : String[?];
    property yoneticiArayuz : ArayuzKay[1];
    invariant KaBileseniIsmiKaIleBaslamali: isim.startsWith('Ka');
}
```

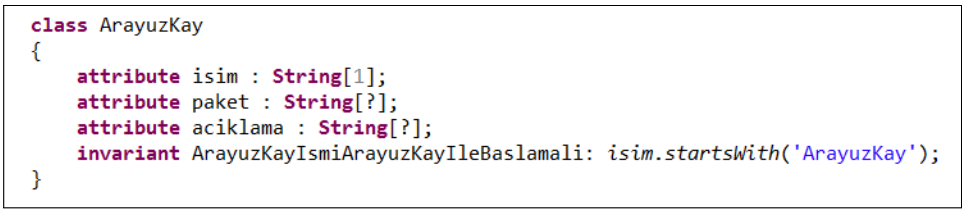
Figure 5.9: OCL KaBileseni element constraint

KaBileseni element constraint shown in Figure 5.9 is that $isim$ attribute value of each KaBileseni element must start with Ka. This rule is expressed in OCL with “KaBileseniIsmiKaIleBaslamali” invariant.

5.10 OCL ArayuzKay Element Constraints

We define a constraint for ArayuzKay elements in a DSS-RSA model. This constraint is shown in Figure 5.10.

ArayuzKay element constraint shown in Figure 5.10 is that $isim$ attribute value of each ArayuzKay element must start with $ArayuzKay$. This rule is expressed in OCL with “ArayuzKayIsmiArayuzKayIleBaslamali” invariant.

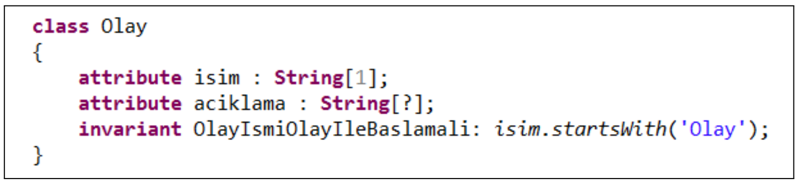


```
class ArayuzKay
{
    attribute isim : String[1];
    attribute paket : String[?];
    attribute aciklama : String[?];
    invariant ArayuzKayIsmiArayuzKayIleBaslamali: isim.startsWith('ArayuzKay');
}
```

Figure 5.10: OCL ArayuzKay element constraint

5.11 OCL Olay Element Constraints

We define a constraint for Olay elements in a DSS-RSA model. This constraint is shown in Figure 5.11.



```
class Olay
{
    attribute isim : String[1];
    attribute aciklama : String[?];
    invariant OlayIsmiOlayIleBaslamali: isim.startsWith('Olay');
}
```

Figure 5.11: OCL Olay element constraint

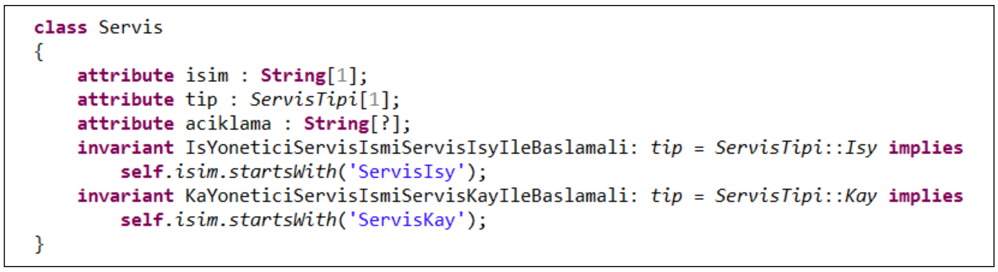
Olay element constraint shown in Figure 5.11 is that $isim$ attribute value of each Olay element must start with $Olay$. This rule is expressed in OCL with “OlayIsmiOlayIle-Baslamali” invariant.

5.12 OCL Servis Element Constraints

We define some constraints for Servis elements in a DSS-RSA model. These constraints are shown in Figure 5.12.

Servis element constraints shown in Figure 5.12 are explained in detail below:

- If a Servis element’s tip attribute value is Isy, $isim$ attribute value of this Servis element must start with $ServisIsy$. This rule is expressed in OCL with “IsYoneticiServisIsmiServisIsyIleBaslamali” invariant.
- If a Servis element’s tip attribute value is Kay, $isim$ attribute value of this



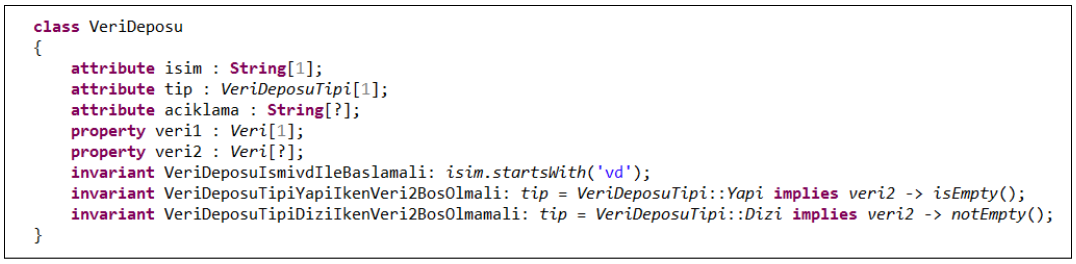
```
class Servis
{
    attribute isim : String[1];
    attribute tip : ServisTipi[1];
    attribute aciklama : String[?];
    invariant IsYoneticiServisIsmiServisIsyIleBaslamali: tip = ServisTipi::Isy implies
        self.isim.startsWith('ServisIsy');
    invariant KaYoneticiServisIsmiServisKayIleBaslamali: tip = ServisTipi::Kay implies
        self.isim.startsWith('ServisKay');
}
```

Figure 5.12: OCL Servis element constraints

Servis element must start with $ServisKay$. This rule is expressed in OCL with "KaYoneticiServisIsmiServisKayIleBaslamali" invariant.

5.13 OCL VeriDeposu Element Constraints

We define some constraints for VeriDeposu elements in a DSS-RSA model. These constraints are shown in Figure 5.13.



```
class VeriDeposu
{
    attribute isim : String[1];
    attribute tip : VeriDeposuTipi[1];
    attribute aciklama : String[?];
    property veri1 : Veri[1];
    property veri2 : Veri[?];
    invariant VeriDeposuIsmivdIleBaslamali: isim.startsWith('vd');
    invariant VeriDeposuTipiYapiIkenVeri2BosOlmali: tip = VeriDeposuTipi::Yapi implies veri2 -> isEmpty();
    invariant VeriDeposuTipiDiziIkenVeri2BosOlmamali: tip = VeriDeposuTipi::Dizi implies veri2 -> notEmpty();
}
```

Figure 5.13: OCL VeriDeposu element constraints

VeriDeposu element constraints shown in Figure 5.13 are explained in detail below:

- $isim$ attribute value of each VeriDeposu element must start with vd. This rule is expressed in OCL with "VeriDeposuIsmivdIleBaslamali" invariant.
- If a VeriDeposu element's tip attribute value is $Yapi$, $veri2$ attribute value must be empty. This rule is expressed in OCL with "VeriDeposuTipiYapiIken-Veri2BosOlmali" invariant.
- If a VeriDeposu element's tip attribute value is $Dizi$, $veri2$ attribute value must

not be empty. This rule is expressed in OCL with "VeriDeposuTipiDiziIken-Veri2BosOlmamali" invariant.

CHAPTER 6

CONCRETE SYNTAX DEFINITION

Based on the DSS-RSA metamodel, we define concrete syntax. Concrete syntax, in general, includes definitions used to create models by using graphical or textual elements. These elements refer to modeling concepts described in metamodel. In the scope of this study, we define a graphical concrete syntax that uses graphical notation to create models. Graphical notation symbolizes a metamodel by introducing symbols for modeling concepts in metamodel as shown in Figure 6.1.

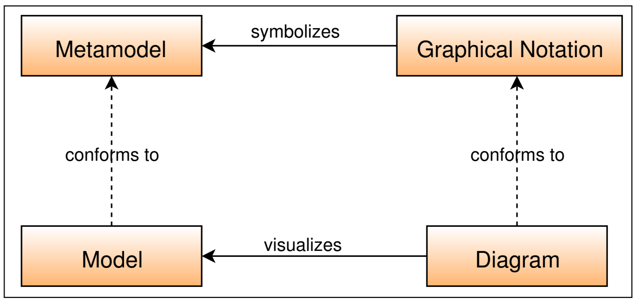


Figure 6.1: Graphical notation introduces symbols for the modeling concepts, adapted from [2]

In Eclipse, Graphical concrete syntax development is supported by the Graphical Modeling Framework (GMF). GMF defines graphical and tooling models to describe the concrete syntax. Then, mapping model is defined to connect elements of the Ecore metamodel to the corresponding elements of the concrete syntax. By developing a generator model based on the mapping model, Java code and other configuration files representing a Domain Specific Language (DSL) are generated. Generated DSL (graphical modeling editor) is run as Eclipse plug-in [16].

Graphical modeling editor is used to manipulate the elements in concrete syntax. By using graphical modeling editor, developers can drag and drop DSS-RSA elements from the palette to the design area, set the attribute values of elements and create associations between elements according to the DSS-RSA metamodel definitions.

GMF consists of two main components which are runtime and tooling. The GMF runtime component provides editor operations such as palette, properties view, toolbars, geometrical shapes, saving a diagram as an image and printing. The GMF tooling component is used to generate graphical modeling editor code. Figure 6.2 shows the GMF-Tooling workflow.

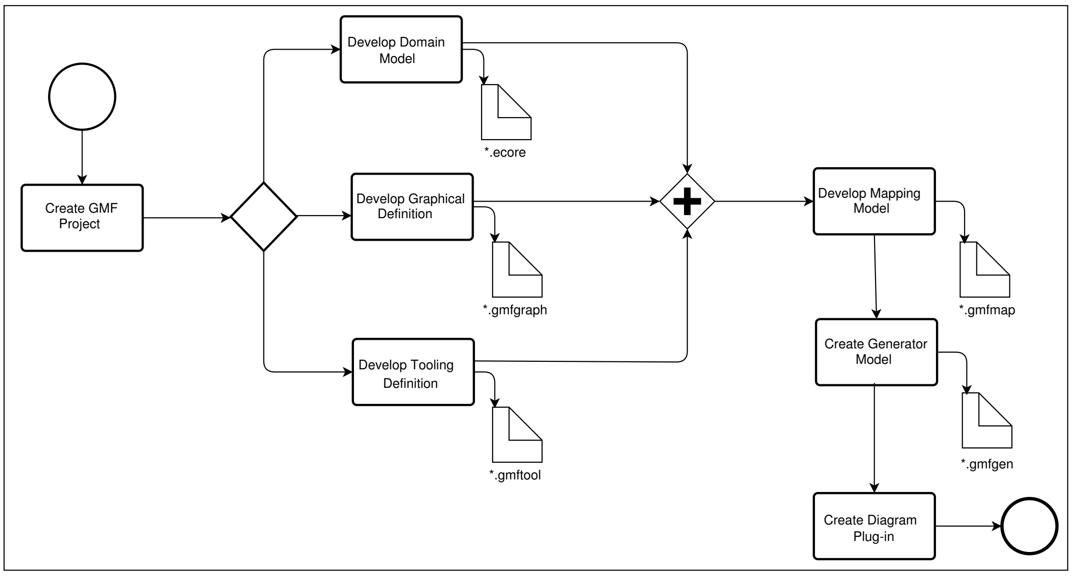


Figure 6.2: GMF-Tooling Workflow, adapted from [3]

In this chapter, models created in GMF-Tooling Workflow are described. Firstly, in Section 6.1 GMF Domain Model, in Section 6.2 EMF Domain Gen Model, in Section 6.3 GMF Graphical Definition Model, in Section 6.4 GMF Tooling Definition Model, in Section 6.5 GMF Mapping Model and finally in Section 6.6 GMF Generator Model is explained. Development steps of graphical modeling editor are given in Appendix B in detail.

6.1 GMF Domain Model

The GMF Domain Model is the DSS-RSA metamodel which is created as EMF Ecore Model and explained in Chapter 4.

6.2 EMF Generator Model

The EMF Generator Model is used for generating EMF code. It allows configuring the properties for code generation that are not part of the Domain Model such as base package name, compliance level and model directory by using properties view in Eclipse [16].

6.3 GMF Graphical Definition Model

The GMF Graphical Definition Model is used to represent GMF Domain Model elements graphically. To do this, GMF Graphical Definition Model allows creating figures that are displayed on the graphical modeling editor and mapping figures with nodes, connections and diagram labels.

The GMF Graphical Definition Model contains a main element whose name is Canvas that has child elements as Figure Gallery, Node, Connection and Diagram Label. Each figure is created by defining a Figure Descriptor element under Figure Gallery. There can be one or more Figure Gallery in GMF Graphical Definition Model. A node can be created by defining a Node element under Canvas. Figure Descriptor related to the Node is set to Node element's Figure property. A connection can be created by defining a Connection element under Canvas. Figure Descriptor related to the Connection is set to Connection element's Figure property. A diagram label can be created by defining a Diagram Label element under Canvas. Figure Descriptor related to the Node that contains this Diagram Label is set to Diagram Label element's Figure property. In the GMF Mapping Model, Canvas elements are used to assign figures to related Domain Model elements.

6.4 GMF Tooling Definition Model

The GMF Tooling Definition Model is used to create diagram palette. To do this, GMF Tooling Definition Model allows creating tool groups and creation tools that are used to create figures on diagram palette. Developers can drag and drop these figures from the palette to the design area.

The GMF Tooling Definition Model contains a main element whose name is Tool Registry that contains Palette as child element. Under Palette element, the user can define Tool Group elements. In this study we define two Tool Groups; one for DSS-RSA elements with the name “DSS RSA Elemanları”, one for DSS-RSA connections with the name “Bağlantılar”. Under Tool Groups, Creation Tool elements are defined. We define Creation Tools related to the DSS-RSA elements under “DSS RSA Elemanları” Tool Group, Creation Tools related to the DSS-RSA connections under “Bağlantılar” Tool Group.

6.5 GMF Mapping Model

The GMF Mapping Model is used to map the Domain Model elements to the related figures that are defined in the GMF Graphical Definition Model and to the related creation tool that are defined in GMF Tooling Definition Model.

The GMF Mapping Model contains a main element with the name Mapping. Mapping element has some child elements but some of them are not used in this study. Top Node Reference, Link Mapping and Canvas Mapping child elements of Mapping element are used. Top Node Reference elements provide mapping of Nodes from the GMF Graphical Definition Model to the related elements of GMF Domain Model and mapping of Creation Tools from the GMF Tooling Definition Model to the related elements of the GMF Domain Model. Link Mapping elements provide mapping of Connections from the GMF Graphical Definition Model to the related connection elements of GMF Domain Model and mapping of Creation Tools from the GMF Tooling Definition Model to the related connection elements of the GMF Domain Model. Canvas Mapping element maps the root element of the GMF Domain Model to the Canvas element from the GMF Graphical Definition Model and to the Palette element from the GMF Tooling Definition Model.

6.6 GMF Generator Model

The GMF Gerenerator Model is used to generate graphical modeling editor code and configuration files by using the GMF Mapping Model and EMF Generator Model. It allows configuring the properties for code generation.

CHAPTER 7

CODE GENERATION

The Model-to-Text transformation is a key concept in Model Driven Engineering. To automate the derivation of text from models by using Model-to-Text transformations, several languages and tools are proposed. They are used for automating some software engineering tasks such as generation of code, test cases, deployment specifications, reports, documents, etc. Code generation is the process of transforming models into source code, which is the most applied Model-to-Text transformation in the field of Model Driven Software Engineering [2].

Eclipse has the Model to Text (M2T) project that focuses on the generation of textual artifacts from models. Jet, Acceleo and Xpand are sub-projects in the M2T project. Xpand is emerged in openArchitectureWare project and migrated to Eclipse as a part of the M2T project [22].

In the scope of this study, a code generation mechanism is implemented using Xpand. Xpand is a statically-typed template language that is based on EMF models and specialized on code generation [22].

In this chapter, code generation mechanism using Xpand template language is described. Firstly, in Section 7.1 creating a new Xpand Project is described. In section 7.2, code generator algorithm for $ArayuzKay\ Elements$ is expained and an example output is given. In section 7.3, code generator algorithm for $KaYonetici\ Elements$ is expained and an example output is given. In section 7.4, code generator algorithm for $KaBileseni\ Elements$ is expained and an example output is given. In section 7.5, code generator algorithm for $IsYonetici\ Elements$ is expained and an example output is given. In section 7.6, code generator algorithm for $IsBileseni\ Elements$

is expained and an example output is given. In section 7.7, code generator algorithm for $Veri\ Elements$ is expained and an example output is given. In section 7.8, code generator algorithm for $Olay\ Elements$ is expained and an example output is given. And finally in section 7.9, code generator algorithm for $Servis\ Elements$ is expained and an example output is given.

7.1 Creating Xpand Project

We create a new Xpand project by clicking on the File menu of Eclipse and selecting "$New \rightarrow Other \rightarrow Xpand\ Project$". We give the project the name "dssml.generator" and select Generate a sample EMF based Xpand project. After clicking Finish, the wizard creates a sample generator project as shown in Figure 7.1.

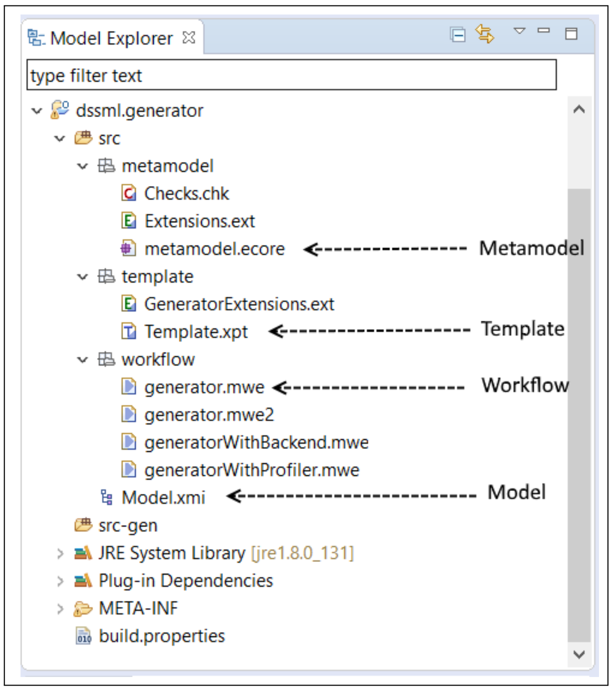


Figure 7.1: Xpand Project

Before we start working with this project, we perform some clean-up actions. We open and delete contents of Checks.chk, Extensions.chk, GeneratorExtensions.ext and Template.xpt files. We write our codes on Template.xpt file. We delete sample metamodel.ecore and Model.xmi files and copy our DSS-RSA metamodel file in metamodel package. After we create a model by using graphical modeling editor described in Chapter 6, we copy the model file under the *src* folder in Xpand project. To generate Java codes under *src-gen* folder, generator.mwe2 file is right-clicked and "$Run\ As \rightarrow MWE2\ Workflow$" is selected.

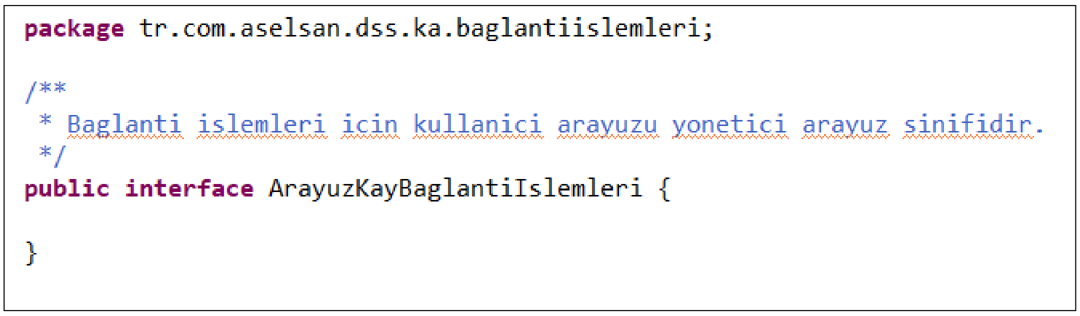
7.2 Code Generator for ArayuzKay Elements

To generate code for ArayuzKay elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for ArayuzKay elements is as follows.

Algorithm 7.2.1 Code Generator Algorithm for ArayuzKay Elements

```
Input: ArayuzKay element list
Output: Java files for each $ArayuzKay\ element$ in ArayuzKay element list with the name represented by $isim$ attribute
 1: for each $ArayuzKay\ element$ ∈ ArayuzKay element list do
 2:     if $paket$ attribute value of $ArayuzKay\ element \neq null\ and\ paket$ attribute value of $ArayuzKay\ element \neq empty$ then
 3:         Declare package by adding $paket$ attribute value of $ArayuzKay\ element$ to the $ka$ package of main package ($anaPaket$)
 4:     else
 5:         Declare package as $ka$ package of main package ($anaPaket$)
 6:     end if
 7:     if $aciklama$ attribute value of $ArayuzKay\ element \neq null\ and\ aciklama$ attribute value of $ArayuzKay\ element \neq empty$ then
 8:         Add interface comment by using $aciklama$ attribute value of $ArayuzKay\ element$
 9:     end if
10:     Declare interface by using $isim$ attribute value of $ArayuzKay\ element$ as name of interface
11: end for
```

Figure 7.2 shows an example code segment which is generated for a ArayuzKay Element.


```java
package tr.com.aselsan.dss.ka.baglantiislemleri;

/**
 * Baglanti islemleri icin kullanici arayuzu yonetici arayuz sinifidir.
 */
public interface ArayuzKayBaglantiIslemleri {

}
```

Figure 7.2: An example code segment generated for a ArayuzKay Element

7.3 Code Generator for KaYonetici Elements

To generate code for KaYonetici elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for KaYonetici elements is shown on Algorithm 7.3.1.

Algorithm 7.3.1 Code Generator Algorithm for KaYonetici Elements

Input: KaYonetici element list, Veri element list

Output: Java files for each $KaYonetici\ element$ in KaYonetici element list with the name represented by $isim$ attribute

```
1: for each $KaYonetici\ element \in$ KaYonetici element list do
2:     if $paket$ attribute value of $KaYonetici\ element \neq null\ and\ paket$ attribute value of $KaYonetici\ element \neq empty$ then
3:         Declare package by adding $paket$ attribute value of $KaYonetici\ element$ to the $ka$ package of main package ($anaPaket$)
4:     else
5:         Declare package as $ka$ package of main package ($anaPaket$)
6:     end if
7:     DeclareImportsForKaYoneticiElement($KaYonetici\ element$, Veri element list)
8:     if $aciklama$ attribute value of $KaYonetici\ element \neq null\ and\ aciklama$ attribute value of $KaYonetici\ element \neq empty$ then
9:         Add class comment by using $aciklama$ attribute value of $KaYonetici\ element$
10:    end if
11:    Declare class by using $isim$ attribute value of $KaYonetici\ element$ as name of class, extending from $KayTemel$ class, implementing interfaces that are $sunulanServis$ relations of $KaYonetici\ element$ and implementing interfaces that are $arayuz$ relations of $KaYonetici\ element$
```

```
12:     Declare class variable $SEVIYE$ as public, static and final and initialize it with the $seviye$ attribute value of $KaYonetici\ element$
13:     Declare class variable $logger$ as private and initialize it
14:     for each $KaBileseni\ element \in ka$ relation of $KaYonetici\ element$ do
15:         Declare class variable for $KaBileseni\ element$ as private by using its $isim$ attribute value
16:     end for
17:     for each $Servis\ element \in kullanilanServis$ relation of $KaYonetici\ element$ do
18:         Declare class variable for $Servis\ element$ as private by using its $isim$ attribute value
19:     end for
20:     Declare a method as protected with the name $edtUzerindeIlklenEleAl$ whose return type is void and add $@Override$ annotation
21:     for each $KaBileseni\ element \in ka$ relation of $KaYonetici\ element$ do
22:         Initialize class variable for $KaBileseni\ element$ in $edtUzerindeIlklenEleAl$ method declaration
23:     end for
24:     for each $Servis\ element \in kullanilanServis$ relation of $KaYonetici\ element$ do
25:         Initialize class variable for $Servis\ element$ by calling $servis$ method and adding a parameter by using $isim$ attribute value of $Servis\ element$ in $edtUzerindeIlklenEleAl$ method declaration
26:     end for
27:     for each $Servis\ element \in sunulanServis$ relation of $KaYonetici\ element$ do
28:         Call $servisEkle$ method by using $isim$ attribute value of $Servis\ element$ in $edtUzerindeIlklenEleAl$ method declaration
29:     end for
30:     Declare a method as protected with the name $edtUzerindeSonlanEleAl$ whose return type is void and add $@Override$ annotation
31:     VeriDeposuSubscriberForKaYoneticiElement ($KaYonetici\ element$, Veri element list)
32:     for each $Olay\ element \in yakalananOlay$ relation of $KaYonetici\ element$ do
33:         Declare a method as private by using $isim$ attribute value of $Olay\ element$, putting $@Abone$ annotation and adding a parameter with name $olay$ and type same with $isim$ attribute value of $Olay\ element$ as final
34:     end for
35: end for
```

The algorithm of code generator for a specific KaYonetici element's Import Declarations is shown on Algorithm 7.3.2.

Algorithm 7.3.2 DeclareImportsForKaYoneticiElement

```
Input: A KaYonetici element, Veri element list
Output: Code segments for import declarations of KaYonetici element
1: Add import declarations for $Logger$ and $LoggerFactory$ classes
2: if $KaYonetici\ element$ has a $aboneVd$ relation with a $VeriDeposu\ element$ whose $tip$ attribute value is $Yapi$ then
3:     Add import declarations for $OlayVeriDeposuYapiIslemi$ and $VeriDeposuYapiIslemiFiltresi$ classes
4: end if
5: if $KaYonetici\ element$ has a $aboneVd$ relation with a $VeriDeposu\ element$ whose $tip$ attribute value is $Dizi$ then
6:     Add import declarations for $OlayVeriDeposuDiziIslemi$ and $VeriDeposuDiziIslemiFiltresi$ classes
7: end if
8: if $aboneVd$ relation of $KaYonetici\ element \neq null$ or $yakalananOlay$ relation of $KaYonetici\ element \neq null$ then
9:     Add import declaration for $Abone$ class
10: end if
11: if $aboneVd$ relation of $KaYonetici\ element \neq null$ then
12:     Add import declarations for $Filtre$ and $ServisIsyVeriDepolari$ classes
13: end if
14: Add import declarations for $KaPanel$ and $KayTemel$ classes
15: for each $Veri\ element \in$ Veri element list do
16:     if $Veri\ element = veri1$ relation of $VeriDeposu\ element$ which is $KaYonetici\ element$'s $aboneVd$ relation or $Veri\ element = veri2$ relation of $VeriDeposu\ element$ which is $KaYonetici\ element$'s $aboneVd$ relation then
17:         Add import declaration for class which is created for $Veri\ element$
18:     end if
19: end for
20: for each $Olay\ element \in yakalananOlay$ relation of $KaYonetici\ element$ do
21:     Add import declaration for class which is created for $Olay\ element$
22: end for
23: for each $Servis\ element \in kullanilanServis$ relation of $KaYonetici\ element$ do
24:     Add import declaration for class which is created for $Servis\ element$
25: end for
26: for each $Servis\ element \in sunulanServis$ relation of $KaYonetici\ element$ do
27:     Add import declaration for class which is created for $Servis\ element$
```

```
28: end for
```

The algorithm of code generator for a specific KaYonetici element's Verideposu Subscriber Methods is shown on Algorithm 7.3.3.

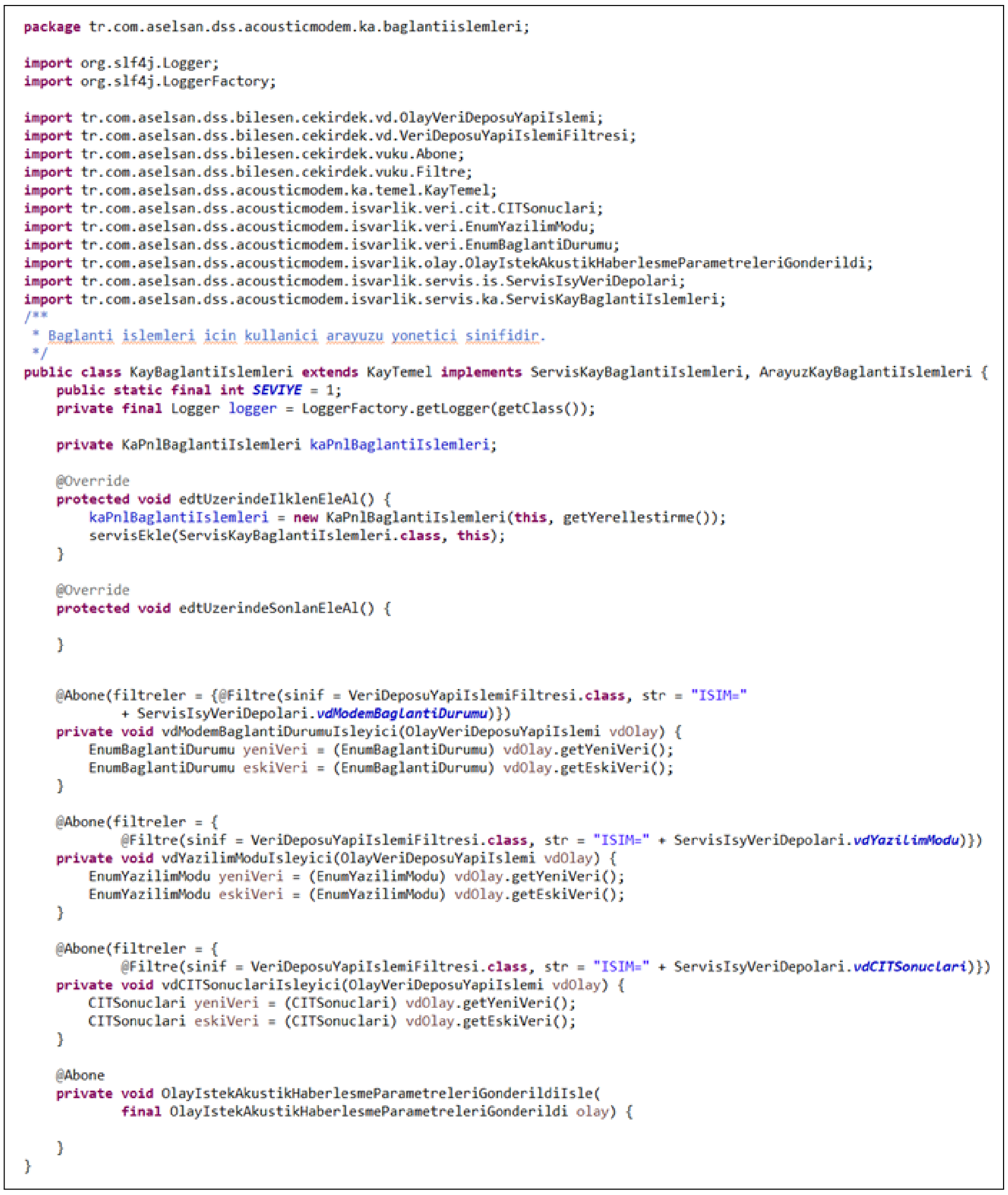
Algorithm 7.3.3 VeriDeposuSubscriberForKaYoneticiElement

Input: A KaYonetici element, Veri element list

Output: Code segments for subscriber methods of VeriDeposu elements which KaYonetici element has $aboneVd$ relation

```
1: for each $VeriDeposu\ element \in aboneVd$ relation of $KaYonetici\ element$ do
2:     if $tip$ attribute value of $VeriDeposu\ element = Yapi$ then
3:         Declare a method as private by using $isim$ attribute value of $VeriDeposu\ element$, putting $@Abone$ annotation and a filter by using $VeriDeposuYapiIslemiFiltresi$ class and adding a parameter with name $vdOlay$ and type $OlayVeriDeposuYapiIslemi$
4:         Declare a variable with the name $yeniVeri$ and initialize it by calling $getYeniVeri$ method of $vdOlay$ parameter
5:         Declare a variable with the name $eskiVeri$ and initialize it by calling $getEskiVeri$ method of $vdOlay$ parameter
6:     else if $tip$ attribute value of $VeriDeposu\ element = Dizi$ then
7:         Declare a method as private by using $isim$ attribute value of $VeriDeposu\ element$, putting $@Abone$ annotation and a filter by using $VeriDeposuDiziIslemiFiltresi$ class and adding a parameter with name $vdOlay$ and type $OlayVeriDeposuDiziIslemi$
8:         Declare a variable with the name $anahtar$ and initialize it by calling $getAnahtar$ method of $vdOlay$ parameter
9:         Declare a variable with the name $yeniVeri$ and initialize it by calling $getYeniVeri$ method of $vdOlay$ parameter
10:    end if
11: end for
```

Figure 7.3 shows an example code segment which is generated for a KaYonetici Element.



```java
package tr.com.aselsan.dss.acousticmodem.ka.baglantiislemleri;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import tr.com.aselsan.dss.bilesen.cekirdek.vd.OlayVeriDeposuYapiIslemi;
import tr.com.aselsan.dss.bilesen.cekirdek.vd.VeriDeposuYapiIslemiFiltresi;
import tr.com.aselsan.dss.bilesen.cekirdek.vuku.Abone;
import tr.com.aselsan.dss.bilesen.cekirdek.vuku.Filtre;
import tr.com.aselsan.dss.acousticmodem.ka.temel.KayTemel;
import tr.com.aselsan.dss.acousticmodem.isvarlik.veri.cit.CITSonuclari;
import tr.com.aselsan.dss.acousticmodem.isvarlik.veri.EnumYazilimModu;
import tr.com.aselsan.dss.acousticmodem.isvarlik.veri.EnumBaglantiDurumu;
import tr.com.aselsan.dss.acousticmodem.isvarlik.olay.OlayIstekAkustikHaberlesmeParametreleriGonderildi;
import tr.com.aselsan.dss.acousticmodem.isvarlik.servis.is.ServisIsyVeriDepolari;
import tr.com.aselsan.dss.acousticmodem.isvarlik.servis.ka.ServisKayBaglantiIslemleri;
/**
 * Baglanti islemleri icin kullanici arayuzu yonetici sinifidir.
 */
public class KayBaglantiIslemleri extends KayTemel implements ServisKayBaglantiIslemleri, ArayuzKayBaglantiIslemleri {
    public static final int SEVIYE = 1;
    private final Logger logger = LoggerFactory.getLogger(getClass());

    private KaPnlBaglantiIslemleri kaPnlBaglantiIslemleri;

    @Override
    protected void edtUzerindeIlklenEleAl() {
        kaPnlBaglantiIslemleri = new KaPnlBaglantiIslemleri(this, getYerellestirme());
        servisEkle(ServisKayBaglantiIslemleri.class, this);
    }

    @Override
    protected void edtUzerindeSonlanEleAl() {

    }


    @Abone(filtreler = {@Filtre(sinif = VeriDeposuYapiIslemiFiltresi.class, str = "ISIM="
            + ServisIsyVeriDepolari.vdModemBaglantiDurumu)})
    private void vdModemBaglantiDurumuIsleyici(OlayVeriDeposuYapiIslemi vdOlay) {
        EnumBaglantiDurumu yeniVeri = (EnumBaglantiDurumu) vdOlay.getYeniVeri();
        EnumBaglantiDurumu eskiVeri = (EnumBaglantiDurumu) vdOlay.getEskiVeri();
    }

    @Abone(filtreler = {
            @Filtre(sinif = VeriDeposuYapiIslemiFiltresi.class, str = "ISIM=" + ServisIsyVeriDepolari.vdYazilimModu)})
    private void vdYazilimModuIsleyici(OlayVeriDeposuYapiIslemi vdOlay) {
        EnumYazilimModu yeniVeri = (EnumYazilimModu) vdOlay.getYeniVeri();
        EnumYazilimModu eskiVeri = (EnumYazilimModu) vdOlay.getEskiVeri();
    }

    @Abone(filtreler = {
            @Filtre(sinif = VeriDeposuYapiIslemiFiltresi.class, str = "ISIM=" + ServisIsyVeriDepolari.vdCITSonuclari)})
    private void vdCITSonuclariIsleyici(OlayVeriDeposuYapiIslemi vdOlay) {
        CITSonuclari yeniVeri = (CITSonuclari) vdOlay.getYeniVeri();
        CITSonuclari eskiVeri = (CITSonuclari) vdOlay.getEskiVeri();
    }

    @Abone
    private void OlayIstekAkustikHaberlesmeParametreleriGonderildiIsle(
            final OlayIstekAkustikHaberlesmeParametreleriGonderildi olay) {

    }
}
```

Figure 7.3: An example code segment generated for a KaYonetici Element

7.4 Code Generator for KaBileseni Elements

To generate code for KaBileseni elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for KaBileseni elements is as follows.

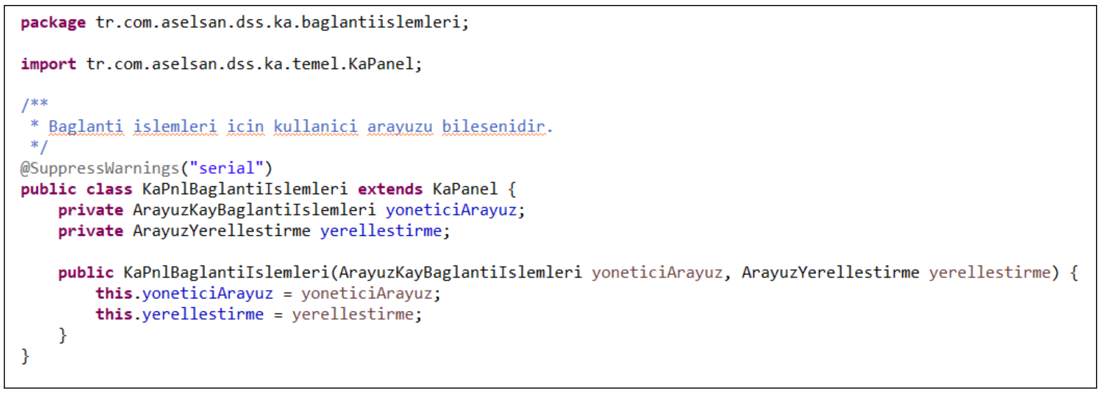
Algorithm 7.4.1 Code Generator Algorithm for KaBileseni Elements

Input: KaBileseni element list

Output: Java files for each $KaBileseni\ element$ in KaBileseni element list with the name represented by $isim$ attribute

```
1: for each $KaBileseni\ element \in$ KaBileseni element list do
2:     if $paket$ attribute value of $KaBileseni\ element \neq null\ and\ paket$ attribute value of $KaBileseni\ element \neq empty$ then
3:         Declare package by adding $paket$ attribute value of $KaBileseni\ element$ to the $ka$ package of main package ($anaPaket$)
4:     else
5:         Declare package as $ka$ package of main package ($anaPaket$)
6:     end if
7:     Add import declaration for $KaPanel$ which will be copied under $temel$ package of $ka$ package
8:     Add import declaration for localization class $ArayuzYerellestirme$ which will be copied under $yerellestirme$ package of $altyapi$ package of project
9:     if $aciklama$ attribute value of $KaBileseni\ element \neq null\ and\ aciklama$ attribute value of $KaBileseni\ element \neq empty$ then
10:        Add class comment by using $aciklama$ attribute value of $KaBileseni\ element$
11:    end if
12:    Add annotation to suppress compiler warnings for serialization
13:    Declare class by using $isim$ attribute value of $KaBileseni\ element$ as name of class and extending from $KaPanel$ class
14:    Declare class variable $yoneticiArayuz$ as private, whose type is the same with $isim$ attribute value of $yoneticiArayuz$ attribute of $KaBileseni\ element$
15:    Declare class variable $yerellestirme$ as private, whose type is $ArayuzYerellestirme$
16:    Add constructor with parameters $yoneticiArayuz$ whose type is the same type with $isim$ attribute value of $yoneticiArayuz$ attribute of $KaBileseni\ element$ and $yerellestirme$ whose type is $ArayuzYerellestirme$
17:    class variable $yoneticiArayuz \leftarrow$ parameter $yoneticiArayuz$
18:    class variable $yerellestirme \leftarrow$ parameter $yerellestirme$
19: end for
```

Figure 7.4 shows an example code segment which is generated for a KaBileseni Element.



```java
package tr.com.aselsan.dss.ka.baglantiislemleri;

import tr.com.aselsan.dss.ka.temel.KaPanel;

/**
 * Baglanti islemleri icin kullanici arayuzu bilesenidir.
 */
@SuppressWarnings("serial")
public class KaPnlBaglantiIslemleri extends KaPanel {
    private ArayuzKayBaglantiIslemleri yoneticiArayuz;
    private ArayuzYerellestirme yerellestirme;

    public KaPnlBaglantiIslemleri(ArayuzKayBaglantiIslemleri yoneticiArayuz, ArayuzYerellestirme yerellestirme) {
        this.yoneticiArayuz = yoneticiArayuz;
        this.yerellestirme = yerellestirme;
    }
}
```

Figure 7.4: An example code segment generated for a KaBileseni Element

7.5 Code Generator for IsYonetici Elements

To generate code for IsYonetici elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for IsYonetici elements is as follows.

Algorithm 7.5.1 Code Generator Algorithm for IsYonetici Elements

Input: IsYonetici element list, Veri element list

Output: Java files for each $IsYonetici\ element$ in IsYonetici element list with the name represented by $isim$ attribute

```
1: for each $IsYonetici\ element \in$ IsYonetici element list do
2:     if $paket$ attribute value of $IsYonetici\ element \neq null\ and\ paket$ attribute value of $IsYonetici\ element \neq empty$ then
3:         Declare package by adding $paket$ attribute value of $IsYonetici\ element$ to the $is$ package of main package ($anaPaket$)
4:     else
5:         Declare package as $is$ package of main package ($anaPaket$)
6:     end if
7:     DeclareImportsForIsYoneticiElement($IsYonetici\ element$, Veri element list)
8:     if $aciklama$ attribute value of $IsYonetici\ element \neq null\ and\ aciklama$ attribute value of $IsYonetici\ element \neq empty$ then
9:         Add class comment by using $aciklama$ attribute value of $IsYonetici\ element$
10:    end if
11:    Declare class by using $isim$ attribute value of $IsYonetici\ element$ as name of class, extending from $YoneticiAdaptor$ class and implementing interfaces that are $sunulanServis$ relations of $IsYonetici\ element$
```

```
12:     Declare class variable $SEVIYE$ as public, static and final and initialize it with the $seviye$
        attribute value of $IsYonetici\ element$
13:     Declare class variable $logger$ as private and initialize it
14:     for each $Servis\ element \in kullanilanServis$ relation of $IsYonetici\ element$ do
15:         Declare a class variable for $Servis\ element$ as private by using its $isim$ attribute value
16:     end for
17:     Declare a method as protected with the name $ilklenEleAl$ whose return type is void and add
        $@Override$ annotation
18:     for each $Servis\ element \in kullanilanServis$ relation of $IsYonetici\ element$ do
19:         Initialize class variable for $Servis\ element$ by calling $servis$ method and adding a param-
            eter by using $isim$ attribute value of $Servis\ element$ in $ilklenEleAl$ method declaration
20:     end for
21:     for each $Servis\ element \in sunulanServis$ relation of $IsYonetici\ element$ do
22:         Call $servisEkle$ method by using $isim$ attribute value of $Servis\ element$ in $ilklenEleAl$
            method declaration
23:     end for
24:     Declare a method as protected with the name $sonlanEleAl$ whose return type is void and add
        $@Override$ annotation
25:     VeriDeposuSubscriberForIsYoneticiElement ($IsYonetici\ element$, Veri element list)
26:     for each $Olay\ element \in yakalananOlay$ relation of $IsYonetici\ element$ do
27:         Declare a method as private by using $isim$ attribute value of $Olay\ element$, putting $@Abo-$
            $ne$ annotation and adding a parameter with name $olay$ and type same with $isim$ attribute
            value of $Olay\ element$ as final
28:     end for
29: end for
```

The algorithm of code generator for a specific IsYonetici element's Import Declarations is shown on Algorithm 7.5.2.

Algorithm 7.5.2 DeclareImportsForIsYoneticiElement

```
Input: A IsYonetici element, Veri element list
Output: Code segments for import declarations of IsYonetici element
1: Add import declarations for $Logger$ and $LoggerFactory$ classes
2: Add import declaration for $YoneticiAdaptor$ class
3: if $IsYonetici\ element$ has a $aboneVd$ relation with a $VeriDeposu\ element$ whose $tip$ attribute
   value is $Yapi$ then
```

```
4:     Add import declarations for $OlayVeriDeposuYapiIslemi$ and $VeriDeposuYapiIslemiFiltresi$ classes
5: end if
6: if $IsYonetici\ element$ has a $aboneVd$ relation with a $VeriDeposu\ element$ whose $tip$ attribute value is $Dizi$ then
7:     Add import declarations for $OlayVeriDeposuDiziIslemi$ and $VeriDeposuDiziIslemiFiltresi$ classes
8: end if
9: if $aboneVd$ relation of $IsYonetici\ element \neq null$ or $yakalananOlay$ relation of $IsYonetici\ element \neq null$ then
10:    Add import declaration for $Abone$ class
11: end if
12: if $aboneVd$ relation of $IsYonetici\ element \neq null$ then
13:    Add import declarations for $Filtre$ and $ServisIsyVeriDepolari$ classes
14: end if
15: for each $Veri\ element \in$ Veri element list do
16:    if $Veri\ element = veri1$ relation of $VeriDeposu\ element$ which is $aboneVd$ relation of $IsYonetici\ element$ or $Veri\ element = veri2$ relation of $VeriDeposu\ element$ which is $aboneVd$ relation of $IsYonetici\ element$ then
17:        Add import declaration for class which is created for $Veri\ element$
18:    end if
19: end for
20: for each $Olay\ element \in yakalananOlay$ relation of $IsYonetici\ element$ do
21:    Add import declaration for class which is created for $Olay\ element$
22: end for
23: for each $Servis\ element \in kullanilanServis$ relation of $IsYonetici\ element$ do
24:    Add import declaration for class which is created for $Servis\ element$
25: end for
26: for each $Servis\ element \in sunulanServis$ relation of $IsYonetici\ element$ do
27:    Add import declaration for class which is created for $Servis\ element$
28: end for
```

The algorithm of code generator for a specific IsYonetici element's Verideposu Subscriber Methods is shown on Algorithm 7.5.1.

Algorithm 7.5.1 VeriDeposuSubscriberForIsYoneticiElement

Input: A IsYonetici element, Veri element list

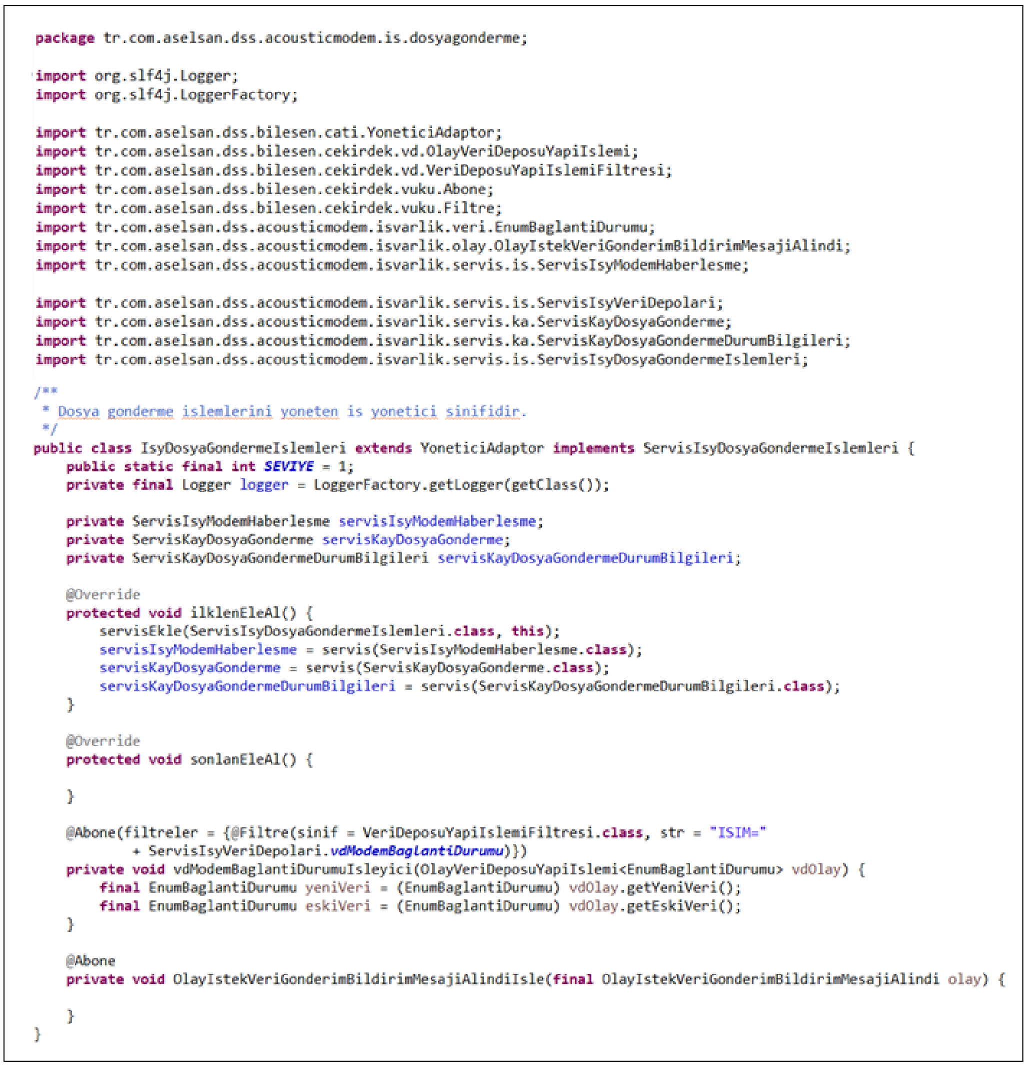
Output: Code segments for subscriber methods of VeriDeposu elements which IsYonetici element has $aboneVd$ relation

```
1:  for each $VeriDeposu\ element \in aboneVd$ relation of $IsYonetici\ element$ do
2:      if $tip$ attribute value of $VeriDeposu\ element = Yapi$ then
3:          Declare a method as private by using $isim$ attribute value of $VeriDeposu\ element$, putting $@Abone$ annotation and a filter by using $VeriDeposuYapiIslemiFiltresi$ class and adding a parameter with name $vdOlay$ and type $OlayVeriDeposuYapiIslemi$
4:          Declare a variable with the name $yeniVeri$ and initialize it by calling $getYeniVeri$ method of $vdOlay$ parameter
5:          Declare a variable with the name $eskiVeri$ and initialize it by calling $getEskiVeri$ method of $vdOlay$ parameter
6:      else if $tip$ attribute value of $VeriDeposu\ element = Dizi$ then
7:          Declare a method as private by using $isim$ attribute value of $VeriDeposu\ element$, putting $@Abone$ annotation and a filter by using $VeriDeposuDiziIslemiFiltresi$ class and adding a parameter with name $vdOlay$ and type $OlayVeriDeposuDiziIslemi$
8:          Declare a variable with the name $anahtar$ and initialize it by calling $getAnahtar$ method of $vdOlay$ parameter
9:          Declare a variable with the name $yeniVeri$ and initialize it by calling $getYeniVeri$ method of $vdOlay$ parameter
10:     end if
11: end for
```

Figure 7.5 shows an example code segment which is generated for a IsYonetici Element.

7.6 Code Generator for IsBileseni Elements

To generate code for IsBileseni elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for IsBileseni elements is given on Algorithm 7.6.1.



```java
package tr.com.aselsan.dss.acousticmodem.is.dosyagonderme;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import tr.com.aselsan.dss.bilesen.cati.YoneticiAdaptor;
import tr.com.aselsan.dss.bilesen.cekirdek.vd.OlayVeriDeposuYapiIslemi;
import tr.com.aselsan.dss.bilesen.cekirdek.vd.VeriDeposuYapiIslemiFiltresi;
import tr.com.aselsan.dss.bilesen.cekirdek.vuku.Abone;
import tr.com.aselsan.dss.bilesen.cekirdek.vuku.Filtre;
import tr.com.aselsan.dss.acousticmodem.isvarlik.veri.EnumBaglantiDurumu;
import tr.com.aselsan.dss.acousticmodem.isvarlik.olay.OlayIstekVeriGonderimBildirimMesajiAlindi;
import tr.com.aselsan.dss.acousticmodem.isvarlik.servis.is.ServisIsyModemHaberlesme;

import tr.com.aselsan.dss.acousticmodem.isvarlik.servis.is.ServisIsyVeriDepolari;
import tr.com.aselsan.dss.acousticmodem.isvarlik.servis.ka.ServisKayDosyaGonderme;
import tr.com.aselsan.dss.acousticmodem.isvarlik.servis.ka.ServisKayDosyaGondermeDurumBilgileri;
import tr.com.aselsan.dss.acousticmodem.isvarlik.servis.is.ServisIsyDosyaGondermeIslemleri;

/**
 * Dosya gonderme islemlerini yoneten is yonetici sinifidir.
 */
public class IsyDosyaGondermeIslemleri extends YoneticiAdaptor implements ServisIsyDosyaGondermeIslemleri {
    public static final int SEVIYE = 1;
    private final Logger logger = LoggerFactory.getLogger(getClass());

    private ServisIsyModemHaberlesme servisIsyModemHaberlesme;
    private ServisKayDosyaGonderme servisKayDosyaGonderme;
    private ServisKayDosyaGondermeDurumBilgileri servisKayDosyaGondermeDurumBilgileri;

    @Override
    protected void ilklenEleAl() {
        servisEkle(ServisIsyDosyaGondermeIslemleri.class, this);
        servisIsyModemHaberlesme = servis(ServisIsyModemHaberlesme.class);
        servisKayDosyaGonderme = servis(ServisKayDosyaGonderme.class);
        servisKayDosyaGondermeDurumBilgileri = servis(ServisKayDosyaGondermeDurumBilgileri.class);
    }

    @Override
    protected void sonlanEleAl() {

    }

    @Abone(filtreler = {@Filtre(sinif = VeriDeposuYapiIslemiFiltresi.class, str = "ISIM="
            + ServisIsyVeriDepolari.vdModemBaglantiDurumu)})
    private void vdModemBaglantiDurumuIsleyici(OlayVeriDeposuYapiIslemi<EnumBaglantiDurumu> vdOlay) {
        final EnumBaglantiDurumu yeniVeri = (EnumBaglantiDurumu) vdOlay.getYeniVeri();
        final EnumBaglantiDurumu eskiVeri = (EnumBaglantiDurumu) vdOlay.getEskiVeri();
    }

    @Abone
    private void OlayIstekVeriGonderimBildirimMesajiAlindiIsle(final OlayIstekVeriGonderimBildirimMesajiAlindi olay) {

    }
}
```

Figure 7.5: An example code segment generated for a IsYonetici Element

Algorithm 7.6.1 Code Generator Algorithm for IsBileseni Elements

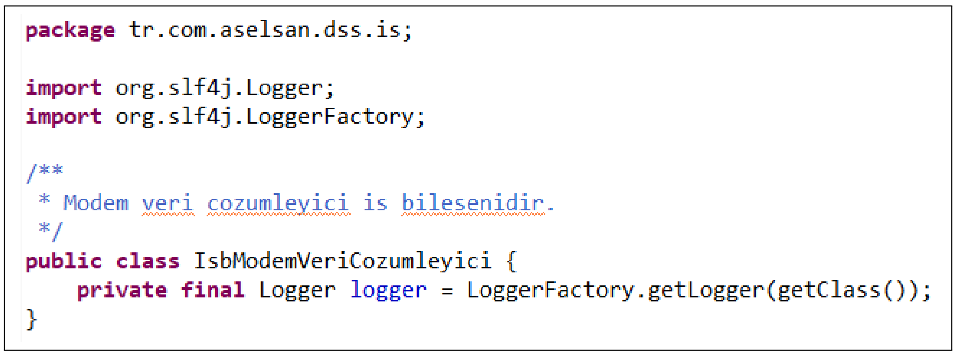
Input: IsBileseni element list

Output: Java files for each $IsBileseni\ element$ in IsBileseni element list with the name represented by $isim$ attribute

```
1: for each $IsBileseni\ element \in$ IsBileseni element list do
2:     if $paket$ attribute value of $IsBileseni\ element \neq null\ and\ paket$ attribute value of $IsBileseni\ element \neq empty$ then
3:         Declare package by adding $paket$ attribute value of $IsBileseni\ element$ to the $is$ package of main package ($anaPaket$)
4:     else
5:         Declare package as $is$ package of main package ($anaPaket$)
```

```
6:     end if
7:     Add import declarations for Logger class
8:     if $aciklama$ attribute value of $IsBileseni\ element \neq null\ and\ aciklama$ attribute value of
       $IsBileseni\ element \neq empty$ then
9:         Add class comment by using $aciklama$ attribute value of $IsBileseni\ element$
10:    end if
11:    Declare class by using $isim$ attribute value of $IsBileseni\ element$ as name of class
12:    Declare class variable $logger$ as private and initialize it
13: end for
```

Figure 7.6 shows an example code segment which is generated for a IsBileseni Element.



```java
package tr.com.aselsan.dss.is;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Modem veri cozumleyici is bilesenidir.
 */
public class IsbModemVeriCozumleyici {
    private final Logger logger = LoggerFactory.getLogger(getClass());
}
```

Figure 7.6: An example code segment generated for a IsBileseni Element

7.7 Code Generator for Veri Elements

To generate code for Veri elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for Veri elements is as follows.

Algorithm 7.7.1 Code Generator Algorithm for Veri Elements

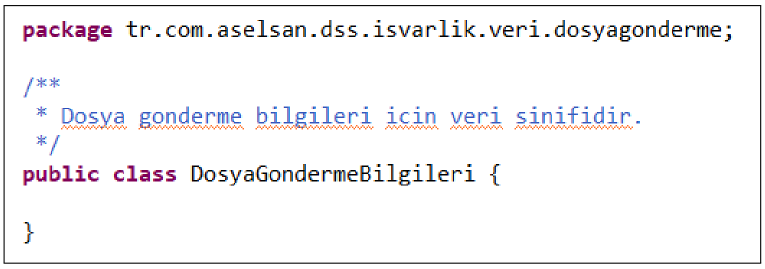
Input: Veri element list

Output: Java files for each $Veri\ element$ in Veri element list with the name represented by $isim$ attribute

```
1: for each $Veri\ element \in$ Veri element list do
2:     if $paket$ attribute value of $Veri\ element \neq null\ and\ paket$ attribute value of $Veri\ element$
       $\neq empty$ then
```

```
3:      Declare package by adding $paket$ attribute value of $Veri\ element$ to the $isvarlik.veri$
        package of main package ($anaPaket$)
4:    else
5:      Declare package as $isvarlik.veri$ package of main package ($anaPaket$)
6:    end if
7:    if $aciklama$ attribute value of $Veri\ element \neq null\ and\ aciklama$ attribute value of $Veri$
      $element \neq empty$ then
8:      Add comment by using $aciklama$ attribute value of $Veri\ element$
9:    end if
10:   if $tip$ attribute value of $Veri\ element$ = $Class$ then
11:     Declare class by using $isim$ attribute value of $Veri\ element$ as name of class
12:   else if $tip$ attribute value of $Veri\ element$ = $Interface$ then
13:     Declare interface by using $isim$ attribute value of $Veri\ element$ as name of interface
14:   else if $tip$ attribute value of $Veri\ element$ = $Enum$ then
15:     Declare enumeration by using $isim$ attribute value of $Veri\ element$ as name of enumera-
        tion
16:   end if
17: end for
```

Figure 7.7 shows an example code segment which is generated for a Veri Element whose tip attribute value is Class. Figure 7.8 shows an example code segment which is generated for a Veri Element whose tip attribute value is Enum.

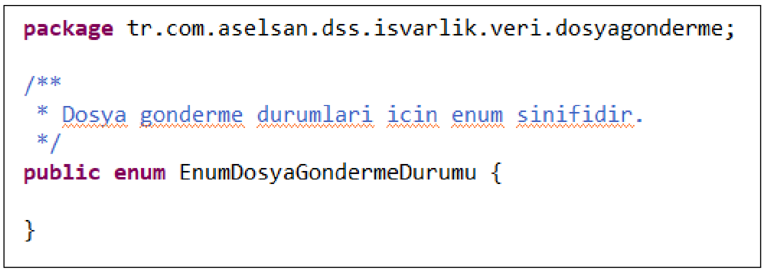


```
package tr.com.aselsan.dss.isvarlik.veri.dosyagonderme;

/**
 * Dosya gonderme bilgileri icin veri sinifidir.
 */
public class DosyaGondermeBilgileri {

}
```

Figure 7.7: An example code segment generated for a Veri Element whose tip attribute value is Class



```
package tr.com.aselsan.dss.isvarlik.veri.dosyagonderme;

/**
 * Dosya gonderme durumlari icin enum sinifidir.
 */
public enum EnumDosyaGondermeDurumu {

}
```

Figure 7.8: An example code segment generated for a Veri Element whose tip attribute value is Enum

7.8 Code Generator for Olay Elements

To generate code for Olay elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for Olay elements is as follows.

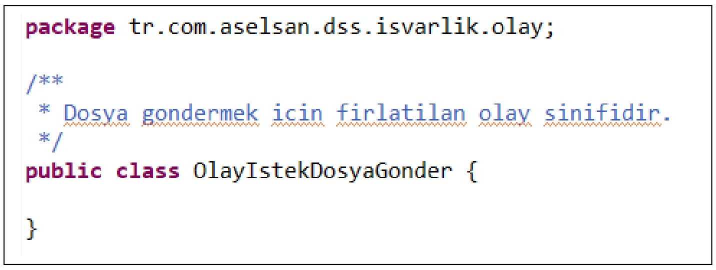
Algorithm 7.8.1 Code Generator Algorithm for Olay Elements

```
Input: Olay element list
Output: Java files for each $Olay\ element$ in Olay element list with the name represented by $isim$
    attribute
1: for each $Olay\ element \in$ Olay element list do
2:     Declare package as $isvarlik.olay$ package of main package ($anaPaket$)
3:     if $aciklama$ attribute value of $Olay\ element \neq null\ and\ aciklama$ attribute value of $Olay$
       $element \neq empty$ then
4:         Add class comment by using $aciklama$ attribute value of $Olay\ element$
5:     end if
6:     Declare class by using $isim$ attribute value of $Olay\ element$ as name of class
7: end for
```

Figure 7.9 shows an example code segment which is generated for a Olay Element.

7.9 Code Generator for Servis Elements

To generate code for Servis elements, we write Xpand codes to the Template.xpt file. The overall algorithm of code generator for Servis elements is as follows.



```
package tr.com.aselsan.dss.isvarlik.olay;

/**
 * Dosya gondermek icin firlatilan olay sinifidir.
 */
public class OlayIstekDosyaGonder {

}
```

Figure 7.9: An example code segment generated for a Olay Element

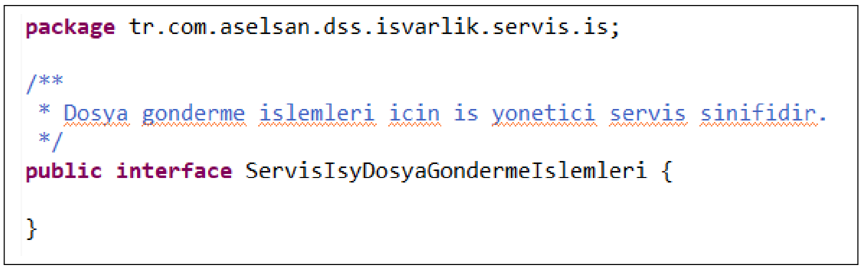
Algorithm 7.9.1 Code Generator Algorithm for Servis Elements

Input: Servis element list

Output: Java files for each $Servis\ element$ in Servis element list with the name represented by $isim$ attribute

```
 1: for each $Servis\ element \in$ Servis element list do
 2:   if $tip$ attribute value of $Servis\ element = Isy$ then
 3:     Declare package as $isvarlik.servis.is$ package of main package ($anaPaket$)
 4:     if $aciklama$ attribute value of $Servis\ element \neq null\ and\ aciklama$ attribute value of
        $Servis\ element \neq empty$ then
 5:       Add interface comment by using $aciklama$ attribute value of $Servis\ element$
 6:     end if
 7:     Declare interface by using $isim$ attribute value of $Servis\ element$ as name of interface
 8:   else if $tip$ attribute value of $Servis\ element = Kay$ then
 9:     Declare package as $isvarlik.servis.ka$ package of main package ($anaPaket$)
10:     if $aciklama$ attribute value of $Servis\ element \neq null\ and\ aciklama$ attribute value of
        $Servis\ element \neq empty$ then
11:       Add interface comment by using $aciklama$ attribute value of $Servis\ element$
12:     end if
13:     Declare interface by using $isim$ attribute value of $Servis\ element$ as name of interface
14:   end if
15: end for
```

Figure 7.10 shows an example code segment which is generated for a Servis Element whose tip attribute value is Isy. Figure 7.11 shows an example code segment which is generated for a Servis Element whose tip attribute value is Kay.

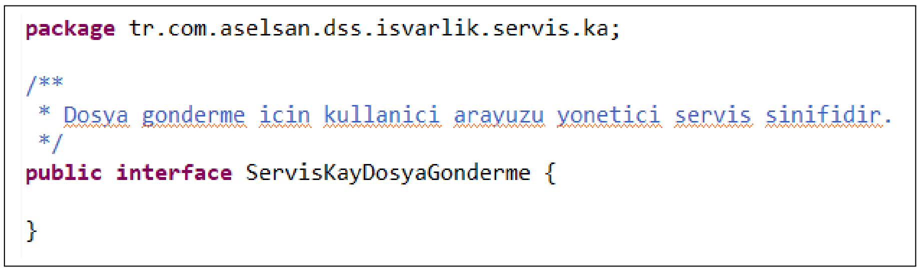


```
package tr.com.aselsan.dss.isvarlik.servis.is;

/**
 * Dosya gonderme islemleri icin is yonetici servis sinifidir.
 */
public interface ServisIsyDosyaGondermeIslemleri {

}
```

Figure 7.10: An example code generated for a Servis Element whose tip attribute value is Isy



```
package tr.com.aselsan.dss.isvarlik.servis.ka;

/**
 * Dosya gonderme icin kullanici arayuzu yonetici servis sinifidir.
 */
public interface ServisKayDosyaGonderme {

}
```

Figure 7.11: An example code generated for a Servis Element whose tip attribute value is Kay

CHAPTER 8

CASE STUDY AND EVALUATION

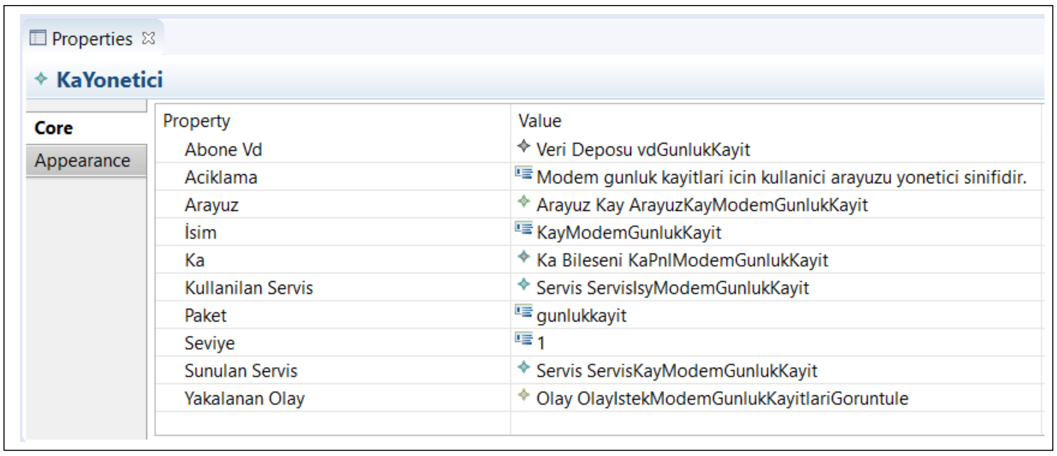
In this chapter, we describe the conducted case study, explain analysis method and give some information about the experimental results.

8.1 Case Study: Acoustic Modem Application Software

In the scope of the case study, Acoustic Modem Application Software has been developed. The software has a graphical user interface that enables the management and monitoring of the modem that provides data exchange over the acoustic environment. Serial communication is established between the modem and the application software. The software allows the modem to transmit files over acoustic environment and records the files sent by another modem on the file system. It displays the daily records of modem, provides user interfaces to adjust communication parameters, does terminal operations, monitors acoustic environment information, starts device test and displays results of test. Graphical user interface presents a toolbar for these actions.

Firstly, a developer who is the author of this thesis implements skeleton codes of Acoustic Modem Application Software manually after architectural design is created. Then, same developer uses proposed Model Driven Engineering approach for implementation of skeleton codes and create application architecture model by using developed graphical modeling editor described in Chapter 6. By using graphical modeling editor, the developer drags and drops DSS-RSA elements from the palette to the design area, set the attribute values of the elements by using Properties View and create associations between the elements according to the DSS-RSA metamodel definitions. Usage of Properties View for a KaYonetici element named as

$KaYoneticiModemGunlukKayit$ to set attribute values is shown in Figure 8.1.


Properties

KaYonetici

Core

Appearance

Property	Value
Abone Vd	Veri Deposu vdGunlukKayit
Aciklama	Modem gunluk kayitlari icin kullanici arayuzu yonetici sinifidir.
Arayuz	Arayuz Kay ArayuzKayModemGunlukKayit
İsim	KayModemGunlukKayit
Ka	Ka Bileseni KaPnlModemGunlukKayit
Kullanilan Servis	Servis ServisIsyModemGunlukKayit
Paket	gunlukkayit
Seviye	1
Sunulan Servis	Servis ServisKayModemGunlukKayit
Yakalanan Olay	Olay OlayIstekModemGunlukKayitlariGoruntule

Figure 8.1: Properties View to edit DSS-RSA elements

Figure 8.2 shows created application architecture model. After application architecture model is created, the model should be validated to check the conformance with the DSS-RSA. As described in Chapter 5, OCL constraints are defined on metamodel by using OCLinEcore editor. To validate the model, design area is right clicked and "$OCL \rightarrow Validate$" is selected as shown in Figure 8.3. If some OCL constraints are violated, validation results show problems. Figure 8.4 shows some validation problems and Figure 8.5 shows successful validation of application architecture model.

After application architecture architecture model is created and validated, the model file, namely Model.xmi, is copied under the *src* folder of Xpand project. To generate skeleton Java codes under *src-gen* folder, generator.mwe2 file is right-clicked and "$Run\ As \rightarrow MWE2\ Workflow$" is selected as decribed in Chapter 7. Generated code examples for each element of DSS-RSA are also given in Chapter 7. Details of generated classes and lines of skeleton codes for each DSS-RSA element are given in Table 8.1.

For this study, development times of two approaches are measured by using a stopwatch and showed in Table 8.2. By using the second approach where proposed method in this thesis is used, expected package structure and skeleton codes are created in a much shorter time than the manual approach. In the first approach developer finished the work in 489 minutes, whereas in the second approach the work took 116 minutes.

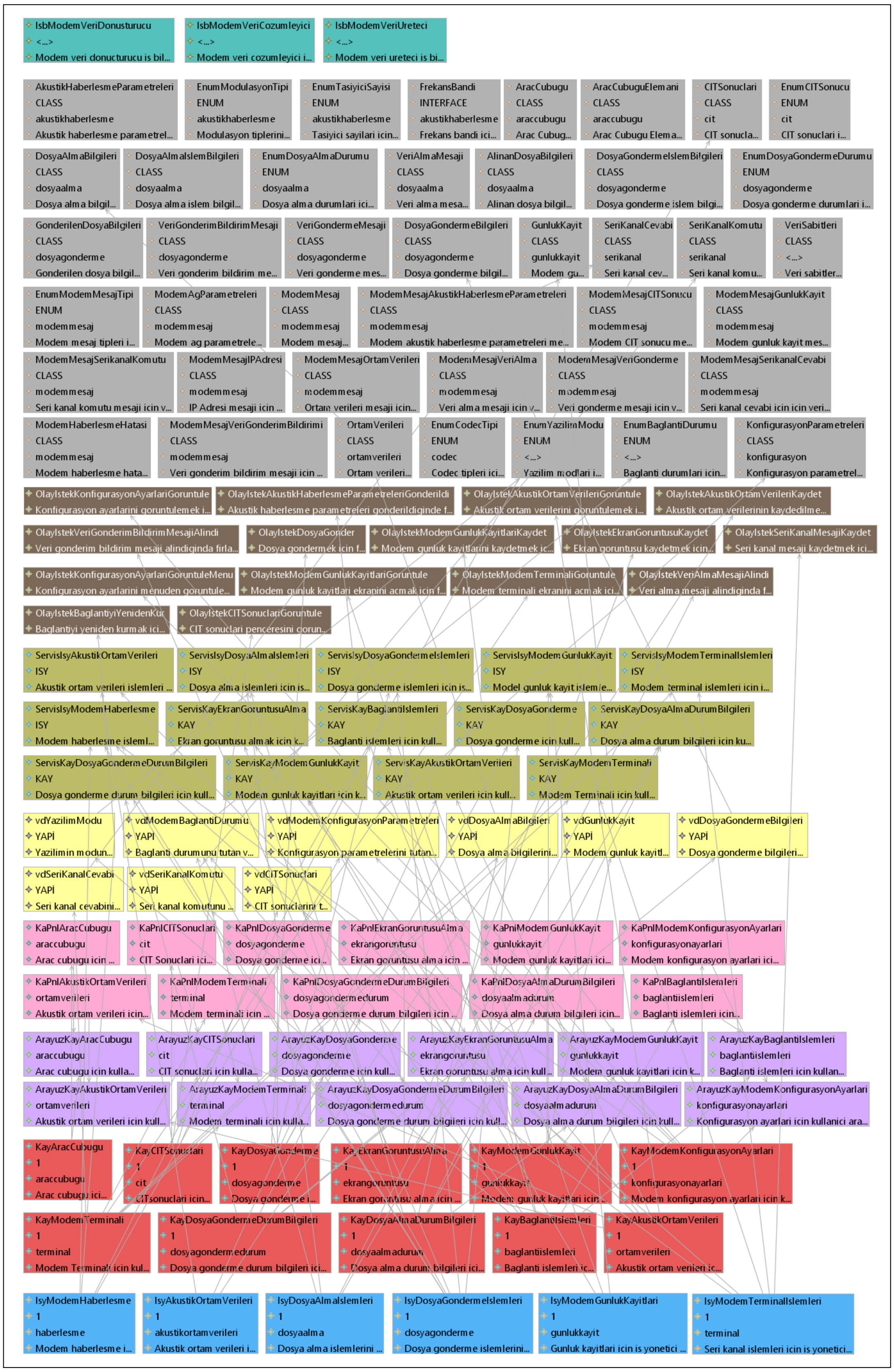

IsbModemVeriDonusturucu <...> Modem veri donuturucu is bil...
IsbModemVeriCozumleyici <...> Modem veri cozumleyici i...
IsbModemVeriUreteci <...> Modem veri ureteci is bi...
AkustikHaberlesmeParametreleri CLASS akustikhaberlesme Akustik haberlesme parametrel...
EnumModulasyonTipi ENUM akustikhaberlesme Modulasyon tiplerini...
EnumTasiyiciSayisi ENUM akustikhaberlesme Tasiyici sayilari icin...
FrekansBandi INTERFACE akustikhaberlesme Frekans bandi ici...
AracCubugu CLASS araccubugu Arac Cubug...
AracCubuguElemani CLASS araccubugu Arac Cubugu Elema...
CITSonuclari CLASS cit CIT sonucla...
EnumCITSonucu ENUM cit CIT sonuclari i...
DosyaAlmaBilgileri CLASS dosyaalma Dosya alma bilgil...
DosyaAlmaIslemBilgileri CLASS dosyaalma Dosya alma islem bilgil...
EnumDosyaAlmaDurumu ENUM dosyaalma Dosya alma durumlari ici...
VeriAlmaMesaji CLASS dosyaalma Veri alma mesa...
AlinanDosyaBilgileri CLASS dosyaalma Alinan dosya bilgil...
DosyaGondermeIslemBilgileri CLASS dosyagonderme Dosya gonderme islem bilgi...
EnumDosyaGondermeDurumu ENUM dosyagonderme Dosya gonderme durumlari i...
GonderilenDosyaBilgileri CLASS dosyagonderme Gonderilen dosya bilgil...
VeriGonderimBildirimMesaji CLASS dosyagonderme Veri gonderim bildirim me...
VeriGondermeMesaji CLASS dosyagonderme Veri gonderme mes...
DosyaGondermeBilgileri CLASS dosyagonderme Dosya gonderme bilgil...
GunlukKayit CLASS gunlukkayit Modem gu...
SeriKanalCevabi CLASS serikanal Seri kanal cev...
SeriKanalKomutu CLASS serikanal Seri kanal komu...
VeriSabitleri CLASS <...> Veri sabitler...
EnumModemMesajTipi ENUM modemmesaj Modem mesaj tipleri i...
ModemAgParametreleri CLASS modemmesaj Modem ag parametrele...
ModemMesaj CLASS modemmesaj Modem mesaj...
ModemMesajAkustikHaberlesmeParametreleri CLASS modemmesaj Modem akustik haberlesme parametreleri me...
ModemMesajCITSonucu CLASS modemmesaj Modem CIT sonucu me...
ModemMesajGunlukKayit CLASS modemmesaj Modem gunluk kayit mes...
ModemMesajSerikanalKomutu CLASS modemmesaj Seri kanal komutu mesaji icin v...
ModemMesajIPAdresi CLASS modemmesaj IP Adresi mesaji icin ...
ModemMesajOrtamVerileri CLASS modemmesaj Ortam verileri mesaji icin...
ModemMesajVeriAlma CLASS modemmesaj Veri alma mesaji icin v...
ModemMesajVeriGonderme CLASS modemmesaj Veri gonderme mesaji icin v...
ModemMesajSerikanalCevabi CLASS modemmesaj Seri kanal cevabi icin icin veri...
ModemHaberlesmeHatasi CLASS modemmesaj Modem haberlesme hata...
ModemMesajVeriGonderimBildirimi CLASS modemmesaj Veri gonderim bildirim mesaji icin ...
OrtamVerileri CLASS ortamverileri Ortam verileri...
EnumCodecTipi ENUM codec Codec tipleri ici...
EnumYazilimModu ENUM <...> Yazilim modlari i...
EnumBaglantiDurumu ENUM <...> Baglanti durumlari icin...
KonfigurasyonParametreleri CLASS konfigurasyon Konfigurasyon parametrel...
OlayIstekKonfigurasyonAyarlariGoruntule Konfigurasyon ayarlarini goruntulemek i...
OlayIstekAkustikHaberlesmeParametreleriGonderildi Akustik haberlesme parametreleri gonderildiginde f...
OlayIstekAkustikOrtamVerileriGoruntule Akustik ortam verilerini goruntulemek i...
OlayIstekAkustikOrtamVerileriKaydet Akustik ortam verilerinin kaydedilme...
OlayIstekVeriGonderimBildirimMesajiAlindi Veri gonderim bildirim mesaji alindiginda firla...
OlayIstekDosyaGonder Dosya gondermek icin f...
OlayIstekModemGunlukKayitlariKaydet Modem gunluk kayitlarini kaydetmek ic...
OlayIstekEkranGoruntusuKaydet Ekran goruntusu kaydetmek icin...
OlayIstekSeriKanalMesajiKaydet Seri kanal mesaji kaydetmek ici...
OlayIstekKonfigurasyonAyarlariGoruntuleMenu Konfigurasyon ayarlarini menuden goruntule...
OlayIstekModemGunlukKayitlariGoruntule Modem gunluk kayitlari ekranini acmak icin f...
OlayIstekModemTerminaliGoruntule Modem terminali ekranini acmak ici...
OlayIstekVeriAlmaMesajiAlindi Veri alma mesaji alindiginda f...
OlayIstekBaglantiyiYenidenKur Baglantiyi yeniden kurmak ici...
OlayIstekCITSonuclariGoruntule CIT sonuclari penceresini gorun...
ServisIsyAkustikOrtamVerileri ISY Akustik ortam verileri islemleri ...
ServisIsyDosyaAlmaIslemleri ISY Dosya alma islemleri icin is...
ServisIsyDosyaGondermeIslemleri ISY Dosya gonderme islemleri icin is...
ServisIsyModemGunlukKayit ISY Model gunluk kayit islemle...
ServisIsyModemTerminalIslemleri ISY Modem terminal islemleri icin i...
ServisIsyModemHaberlesme ISY Modem haberlesme isleml...
ServisKayEkranGoruntusuAlma KAY Ekran goruntusu almak icin k...
ServisKayBaglantiIslemleri KAY Baglanti islemleri icin kull...
ServisKayDosyaGonderme KAY Dosya gonderme icin kull...
ServisKayDosyaAlmaDurumBilgileri KAY Dosya alma durum bilgileri icin ku...
ServisKayDosyaGondermeDurumBilgileri KAY Dosya gonderme durum bilgileri icin kull...
ServisKayModemGunlukKayit KAY Modem gunluk kayitlari icin k...
ServisKayAkustikOrtamVerileri KAY Akustik ortam verileri icin kull...
ServisKayModemTerminali KAY Modem Terminali icin kull...
vdYazilimModu YAPI Yazilimin modun...
vdModemBaglantiDurumu YAPI Baglanti durumunu tutan v...
vdModemKonfigurasyonParametreleri YAPI Konfigurasyon parametrelerini tutan...
vdDosyaAlmaBilgileri YAPI Dosya alma bilgilerini...
vdGunlukKayit YAPI Modem gunluk kayitl...
vdDosyaGondermeBilgileri YAPI Dosya gonderme bilgileri...
vdSeriKanalCevabi YAPI Seri kanal cevabini...
vdSeriKanalKomutu YAPI Seri kanal komutunu ...
vdCITSonuclari YAPI CIT sonuclarini t...
KaPnlAracCubugu araccubugu Arac cubugu icin ...
KaPnlCITSonuclari cit CIT Sonuclari ici...
KaPnlDosyaGonderme dosyagonderme Dosya gonderme ici...
KaPnlEkranGoruntusuAlma ekrangoruntusu Ekran goruntusu alma icin ...
KaPnlModemGunlukKayit gunlukkayit Modem gunluk kayitlari ici...
KaPnlModemKonfigurasyonAyarlari konfigurasyonayarlari Modem konfigurasyon ayarlari ici...
KaPnlAkustikOrtamVerileri ortamverileri Akustik ortam verileri icin...
KaPnlModemTerminali terminal Modem terminali icin ...
KaPnlDosyaGondermeDurumBilgileri dosyagondermedurum Dosya gonderme durum bilgileri icin ...
KaPnlDosyaAlmaDurumBilgileri dosyaalmadurum Dosya alma durum bilgileri icin...
KaPnlBaglantiIslemleri baglantiislemleri Baglanti islemleri icin...
ArayuzKayAracCubugu araccubugu Arac cubugu icin kulla...
ArayuzKayCITSonuclari cit CIT sonuclari icin kulla...
ArayuzKayDosyaGonderme dosyagonderme Dosya gonderme icin kull...
ArayuzKayEkranGoruntusuAlma ekrangoruntusu Ekran goruntusu alma icin kull...
ArayuzKayModemGunlukKayit gunlukkayit Modem gunluk kayitlari icin k...
ArayuzKayBaglantiIslemleri baglantiislemleri Baglanti islemleri icin kullan...
ArayuzKayAkustikOrtamVerileri ortamverileri Akustik ortam verileri icin kull...
ArayuzKayModemTerminali terminal Modem terminali icin kulla...
ArayuzKayDosyaGondermeDurumBilgileri dosyagondermedurum Dosya gonderme durum bilgileri icin kull...
ArayuzKayDosyaAlmaDurumBilgileri dosyaalmadurum Dosya alma durum bilgileri icin kull...
ArayuzKayModemKonfigurasyonAyarlari konfigurasyonayarlari Konfigurasyon ayarlari icin kullanici ara...
KayAracCubugu 1 araccubugu Arac cubugu ici...
KayCITSonuclari 1 cit CITsonuclari icin...
KayDosyaGonderme 1 dosyagonderme Dosya gonderme i...
KayEkranGoruntusuAlma 1 ekrangoruntusu Ekran goruntusu alma icin ...
KayModemGunlukKayit 1 gunlukkayit Modem gunluk kayitlari icin ...
KayModemKonfigurasyonAyarlari 1 konfigurasyonayarlari Modem konfigurasyon ayarlari icin k...
KayModemTerminali 1 terminal Modem Terminali icin kul...
KayDosyaGondermeDurumBilgileri 1 dosyagondermedurum Dosya gonderme durum bilgileri ici...
KayDosyaAlmaDurumBilgileri 1 dosyaalmadurum Dosya alma durum bilgileri ici...
KayBaglantiIslemleri 1 baglantiislemleri Baglanti islemleri ic...
KayAkustikOrtamVerileri 1 ortamverileri Akustik ortam verileri ic...
IsyModemHaberlesme 1 haberlesme Modem haberlesme i...
IsyAkustikOrtamVerileri 1 akustikortamverileri Akustik ortam verileri i...
IsyDosyaAlmaIslemleri 1 dosyaalma Dosya alma islemlerini ...
IsyDosyaGondermeIslemleri 1 dosyagonderme Dosya gonderme islemlerini...
IsyModemGunlukKayitlari 1 gunlukkayit Gunluk kayitlari icin is yonetici ...
IsyModemTerminalIslemleri 1 terminal Seri kanal islemleri icin is yonetici...

Figure 8.2: Application model for Acoustic Modem Application Software

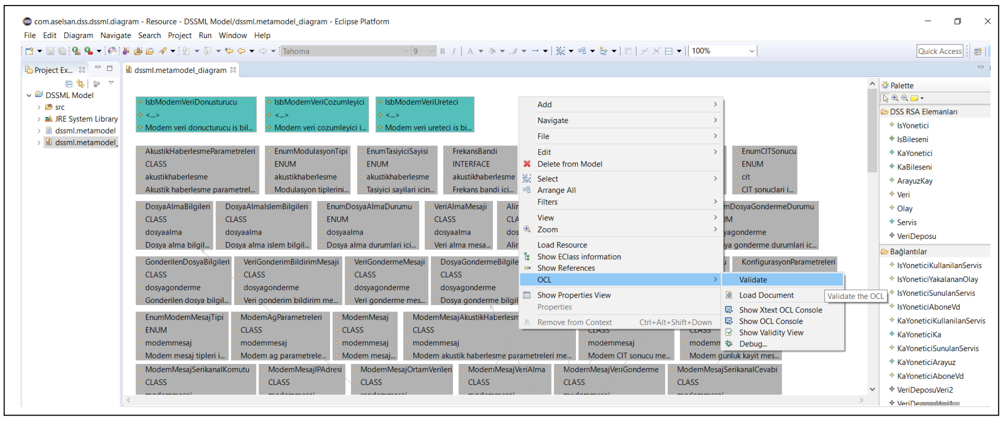
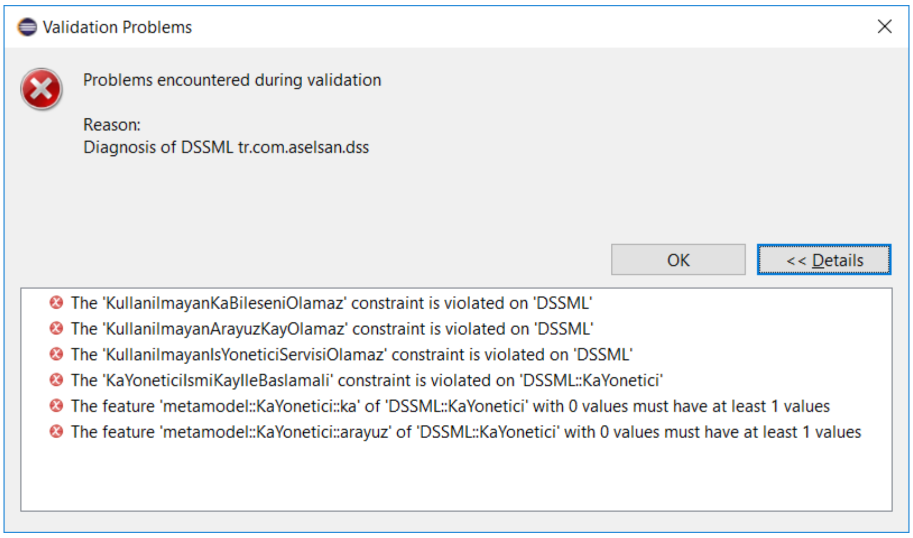


Figure 8.3: Validating application architecture model

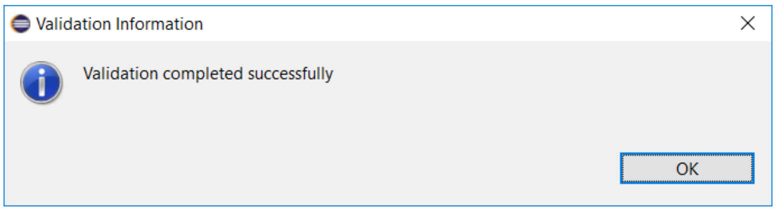

Validation Problems

Problems encountered during validation

Reason:
Diagnosis of DSSML tr.com.aselsan.dss

The 'KullanilmayanKaBileseniOlamaz' constraint is violated on 'DSSML'
The 'KullanilmayanArayuzKayOlamaz' constraint is violated on 'DSSML'
The 'KullanilmayanIsYoneticiServisiOlamaz' constraint is violated on 'DSSML'
The 'KaYoneticiIsmiKayIleBaslamali' constraint is violated on 'DSSML::KaYonetici'
The feature 'metamodel::KaYonetici::ka' of 'DSSML::KaYonetici' with 0 values must have at least 1 values
The feature 'metamodel::KaYonetici::arayuz' of 'DSSML::KaYonetici' with 0 values must have at least 1 values

Figure 8.4: Validation problems of the application architecture model


Validation Information

Validation completed successfully

Figure 8.5: Successful validation of the application architecture model

In addition to that, source code is examined in code review by another developer in DSS Team for compliance with the DSS-RSA. In first approach, some cases of non-compliance with Reference Software Architecture is detected and listed below.

Table 8.1: Generation details for each element of DSS-RSA

DSS-RSA Element	**Class/Enum/Interface Count**	**Package Count**	**Lines of Code**
Veri	9 Enum 33 Class	11	216
Olay	15 Class	-	90
Servis (tip=Isy)	6 Interface	-	36
Servis (tip=Kay)	8 Interface	-	48
ArayuzKay	11 Interface	-	66
KaBileseni	11 Class	-	168
KaYonetici	11 Class	11	395
IsYonetici	6 Class	6	210
IsBileseni	3 Class	0	27
VeriDeposu	1 Class 1 Interface	1	134
Total	**26 Interface** **9 Enum** **80 Class**	**29 Package**	**1390 LOC**

- A naming rule for KaYonetici element of DSS-RSA is violated. A KaYonetici element named as $KayEkranGoruntusuAlma$ is written as $KAYEkranGoruntusuAlma$ mistakenly. This rule can not be violated in the second approach, because it is validated via OCL with $KaYoneticiIsmiKayIleBaslamali$ invariant.

- A naming rule for IsYonetici element of DSS-RSA is violated. A IsYonetici element named as $IsyModemHaberlesme$ is written as $IsYModemHaberlesme$ mistakenly. This rule can not be violated in the second approach, because it is validated via OCL with $IsYoneticiIsmiIsyIleBaslamali$ invariant.

- An element usage rule of DSS-RSA is violated. An Interface named as $ServisKayAracCubugu$ is created for a Servis element whose tip attribute value is Kay but it is not implemented by any KaYonetici element. In other words, there is no $sunulanServis$ relationship between a KaYonetici element and this Servis element. This rule can not be violated in the second approach, because

Table 8.2: Manual and Automatic Development (Proposed Approach) Times for each element of DSS-RSA

DSS-RSA Element	**Manual Development Time (min.)**	**Automatic Development Time (min.)**
Veri	78	30
Olay	37	15
Servis (tip=Isy)	13	6
Servis (tip=Kay)	22	11
ArayuzKay	27	10
KaBileseni	60	11
KaYonetici	105	12
IsYonetici	75	9
IsBileseni	10	3
VeriDeposu	62	9
Total	**489 minutes**	**116 minutes**

it is validated via OCL with $KullanilmayanKaYoneticiServisiOlamaz$ invariant.

- A KaYonetici element rule of DSS-RSA is violated. KaBileseni element named as $KaPnlBaglantiIslemleri$ do not have the same $paket$ attribute value with a KaYonetici element named as $KayBaglantiIslemleri$ which has ka relationship with this KaBileseni element. $paket$ attribute value of KaBileseni element is $baglanti$ but $paket$ attribute value of KaYonetici element is $baglanti$-$islemleri$. This rule can not be violated in the second approach, because it is validated via OCL with $KaYoneticiVeKaBileseniPaketIsimleriAyni$-$Olmali$ invariant.

In the second approach, no case of non-compliance with Reference Software Architecture is detected. Proposed approach in this thesis increases the correctness of the software. Units that are non-compliant with Reference Software Architecture possibly cause errors in software development lifecycle. Creating elements by using graphical modeling editor is easy and it takes short time. Developing code manually takes a long time and is error-prone.

As a case study, a small example consisting of only one subsystem was developed by a single developer. Thus further process data needs to be collected and analyzed to measure the true effect of the proposed approach.

CHAPTER 9

CONCLUSION

In this thesis, we propose a Model Driven Engineering approach to enforce the Reference Software Architecture and to aid in the transition process from architectural design of application software, which is compliant with a Reference Software Architecture, to implementation. Reference Software Architecture provides architectural best practices that are gathered from past experiences to all project team members. They can be standards, prior project artifacts, design patterns, commercial frameworks, and so forth. Reference Software Architecture provides tried and true repeatable processes and reduce the likelihood of incorrect technology decisions. Identification of a Reference Software Architecture can lead to a faster and more reliable software development process [23].

Our approach is realized in three stages by utilizing Model Driven Engineering techniques and tools. In the first stage, we formalize the Reference Software Architecture, namely DSS-RSA, as a metamodel, which is created by DSS Software Team. Reference Software Architecture metamodeling process is started out with elements of DSS-RSA and relationships between these elements. Eclipse Modeling Framework is used for Ecore-based metamodel definition. After DSS-RSA Metamodel is created, OCL constraints for this metamodel are defined by using OCLinEcore Editor. Metamodel can express only very basic modeling constraints, for example cardinality constraints, association ends and types for attributes. More complicated constraints can only be expressed as static semantic rules by using OCL. In the second stage, based on the DSS-RSA Metamodel, concrete syntax is defined by using Eclipse Graphical Modeling Framework. At the end of the concrete syntax definition, a graphical modeling editor is developed. Architectural designs of all application software in

DSS Software Team are created through this graphical modeling editor to promote the Reference Software Architecture compliance of architectural design. Finally, in the third stage, a code generation mechanism is implemented using Xpand which is a statically-typed template language that is based on EMF models and specialized on code generation [22]. By using model file which is the output of the graphical modeling editor and automatic code generation mechanism, skeleton codes are generated. Thus, transition process from architectural design to implementation is facilitated. Furthermore, development of software in compliance with the Reference Software Architecture is promoted.

In this thesis, a case study involving Acoustic Modem Application Software is conducted. The software has a graphical user interface that enables the management and monitoring of the modem that provides data exchange over the acoustic environment. A developer who is the author of this thesis implements the skeleton codes of this application software manually after architectural design is done. Then, same developer uses the proposed method for implementation and creates application architecture model by using graphical modeling editor. Development times of two approaches are measured. By applying proposed method, expected package structure and skeleton codes are created in a much shorter time than the manual approach. In the first approach, the developer finished the implementation of skeleton codes in 489 minutes while the second approach took 116 minutes to build the application architecture model and generate skeleton codes. Therefore, by automatic generation of skeleton codes, savings are achieved in labor. In addition to that, source code is examined in code review by another developer in DSS Software Team for compliance with the Reference Software Architecture. In the first approach, some cases of non-compliance with Reference Software Architecture is detected. But in the second approach, no case of non-compliance with Reference Software Architecture is detected.

To conclude, proposed approach in this thesis increases the correctness of the software by promoting the Reference Software Architecture compliance of architectural design. Moreover, it prevents architecture erosion problem on a large scale and shortens development time. Creating elements by using graphical modeling editor is easy and it takes short time, whereas developing code manually for takes more time and is error-prone. The case study is obviously quite limited in scope, yet promising in regards to long term benefits for the Sea Defense System Software Team.

In the future, we plan to automatically generate the architectural design documents. We can also compare graphical modeling with textual modeling by using results of the study presented in [11].

REFERENCES

[1] M. L. Bernardi, G. A. D. Lucca, and D. Distante, "A model-driven approach for the fast prototyping of web applications," in *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*, pp. 65–74, Sept 2011.

[2] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 1st ed., 2012.

[3] S. Bouchet, *Graphical Modeling Framework/Tutorial/Part 1*, 2013 (last accessed July 8, 2017). available: `https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1`.

[4] E. Y. Nakagawa, P. Oliveira Antonino, and M. Becker, *Reference Architecture and Product Line Architecture: A Subtle But Critical Difference*, pp. 207–211. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[5] D. Quintero, *IBM software defined environment.* IBM redbooks, Poughkeepsie, NY : IBM Corporation, International Technical Support Organization, 2015., 2015.

[6] M. Panunzio, "Definition, realization and evaluation of a software reference architecture for use in space applications (ph.d. thesis)," Tech. Rep. UBLCS-2011-07, University of Bologna (Italy). Department of Computer Science, July 2011.

[7] V. V. Graciano Neto, L. Garcés, M. Guessi, L. B. R. de Oliveira, and F. Oquendo, "On the equivalence between reference architectures and metamodels," in *Proceedings of the 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures*, CobRA '15, (New York, NY, USA), pp. 21–24, ACM, 2015.

[8] S. Martínez-Fernández, C. Ayala, X. Franch, D. Ameller, and H. M. Marques, "A framework for software reference architecture analysis and review.," in *CIbSE 2013: 16th Ibero-American Conference on Software Engineering -*

Memorias del 10th Workshop Latinoamericano Ingenieria de Software Experimental, ESELAW 2013, (GESSI Research Group, Universitat Politécnica de Catalunya), pp. 89–102, 2013.

[9] B. Tekinerdogan, “Chapter 10: Architectural drift analysis using architecture reflexion viewpoint and design structure reflexion matrices.,” *Software Quality Assurance*, pp. 221 – 236, 2016.

[10] J. Rosik, A. Le Gear, J. Buckley, M. A. Babar, and D. Connolly, “Assessing architectural drift in commercial software development: a case study,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 63–86, 2011.

[11] N. K. Turhan and H. Oğuztüzün, “Metamodeling of reference software architecture and automatic code generation,” in *Proccedings of the 10th European Conference on Software Architecture Workshops*, ECSAW ’16, (New York, NY, USA), pp. 2:1–2:7, ACM, 2016.

[12] P. Trojanek, “Model-driven engineering approach to design and implementation of robot control system,” *CoRR*, vol. abs/1302.5085, February 2013.

[13] D. Altunbay, E. Çetinkaya, and M. G. Metin, “Model driven development of board games,” *First Turkish Symposium on Model-Driven Software Development (TMODELS)*, May 2009.

[14] M. L. Bernardi, G. A. D. Lucca, and D. Distante, “Model-driven fast prototyping of rias: From conceptual models to running applications,” in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 250–258, Sept 2014.

[15] H. B. Saritas and G. Kardas, “A model driven architecture for the development of smart card software,” *Computer Languages, Systems & Structures*, vol. 40, no. 2, pp. 53 – 72, 2014.

[16] M. S. Eloumri, *Graphical editors generation with the graphical modeling framework: A case study*. PhD thesis, Queen’s University, 2011.

[17] C. Miksovic and O. Zimmermann, “Architecturally significant requirements, reference architecture, and metamodel for knowledge management in information technology services,” in *Proceedings of the 2011 Ninth Working IEEE/IFIP*

Conference on Software Architecture, WICSA '11, (Washington, DC, USA), pp. 270–279, IEEE Computer Society, 2011.

[18] M. Patterns, *Microsoft Application Architecture Guide*. Microsoft Press, 2nd ed., 2009.

[19] *OCL Specification*, 2014 (last accessed May 27, 2017). available: `http://www.omg.org/spec/OCL/2.4/`.

[20] *Eclipse OCL (Object Constraint Language)*, 2013 (last accessed May 27, 2017). available: `https://projects.eclipse.org/projects/modeling.mdt.ocl`.

[21] E. Willink, *OCL/OCLinEcore*, 2013 (last accessed May 27, 2017). available: `https://wiki.eclipse.org/OCL/OCLinEcore`.

[22] N. Skrypuch, *Eclipse Model To Text (M2T) Project*, last accessed July 16, 2017. available: `https://eclipse.org/modeling/m2t`.

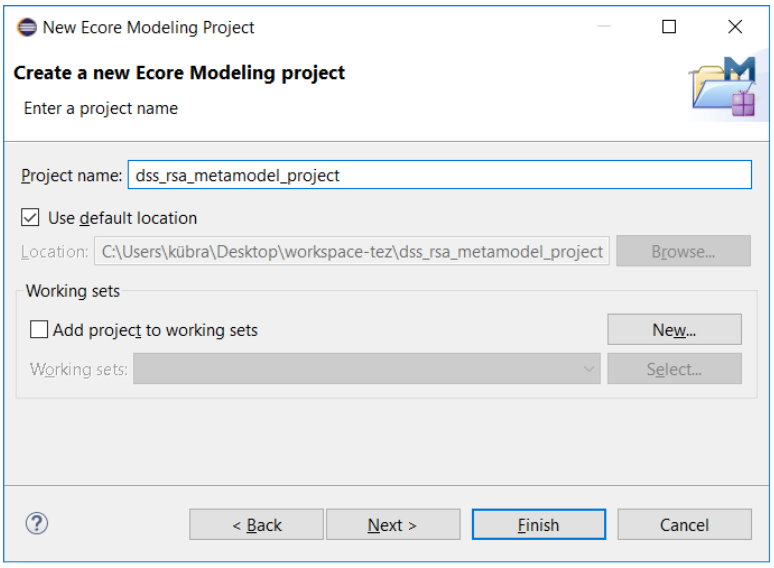
[23] P. Reed, *Reference Architecture: The best of best practices*, 2002 (last accessed August 12, 2017). available: `https://www.ibm.com/developerworks/rational/library/2774.html`.

APPENDIX A

DEVELOPING DOMAIN MODEL AS EMF ECORE MODEL

A.1 Creating EMF Project

We create a new project to hold domain model by clicking on the File menu of Eclipse and selecting “$New \rightarrow Other \rightarrow Ecore\ Modeling\ Project$”. We give the project the name “dss_rsa_metamodel_project” as shown in Figure A.1. Then, we define the model settings as shown in Figure A.2 and select the viewpoints to activate as shown in Figure A.3. After clicking finish, Ecore Modeling Project is created.


New Ecore Modeling Project

Create a new Ecore Modeling project

Enter a project name

Project name: dss_rsa_metamodel_project

☑ Use default location

Location: C:\Users\kübra\Desktop\workspace-tez\dss_rsa_metamodel_project

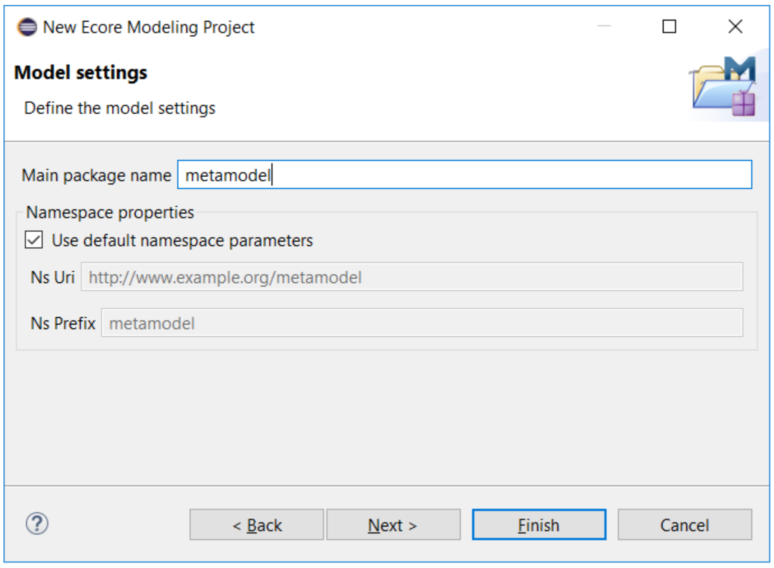
Working sets

☐ Add project to working sets

Working sets:

Figure A.1: Creating new Ecore Modeling Project: enter a project name

By using the created diagram shown in Figure A.4, elements are dragged&dropped from Palette into the design area and Domain Model of DSS-RSA is created as shown in Figure A.5.


New Ecore Modeling Project

Model settings

Define the model settings

Main package name: metamodel

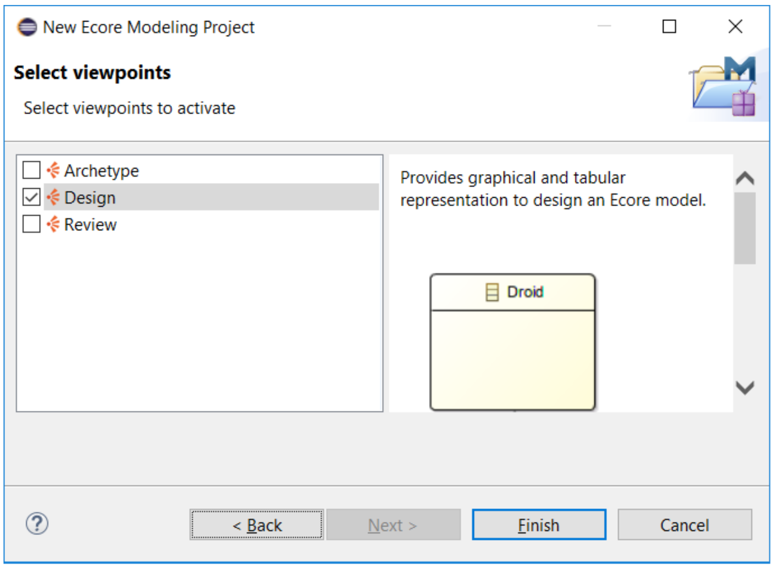
Namespace properties

☑ Use default namespace parameters

Ns Uri: http://www.example.org/metamodel

Ns Prefix: metamodel

Figure A.2: Creating new Ecore Modeling Project: define the model settings


New Ecore Modeling Project

Select viewpoints

Select viewpoints to activate

☐ Archetype
☑ Design
☐ Review

Provides graphical and tabular representation to design an Ecore model.

Figure A.3: Creating new Ecore Modeling Project: select viewpoints to activate

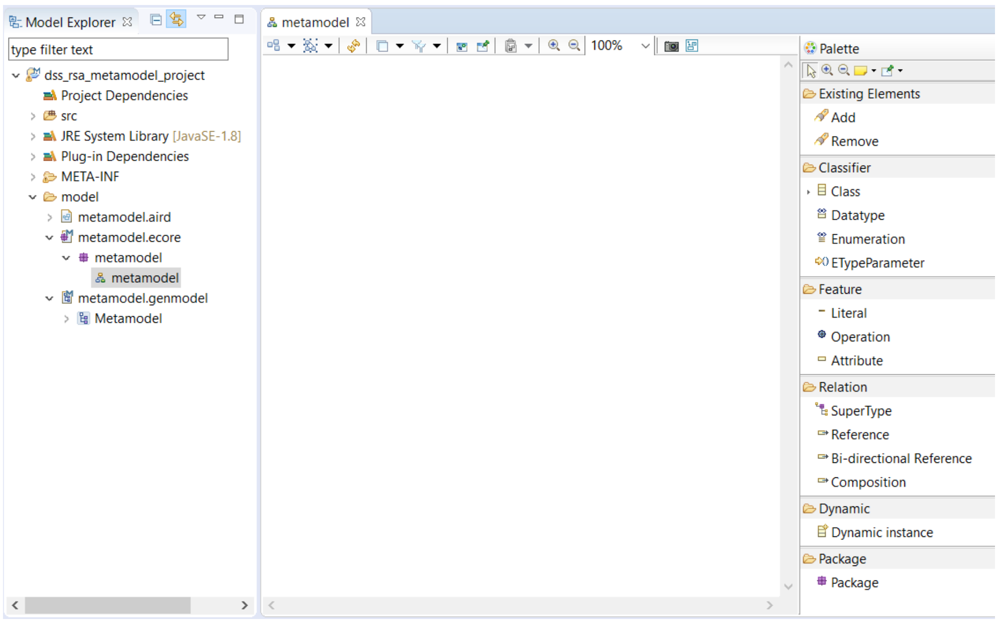


Figure A.4: Design area and palette of Ecore modeling editor

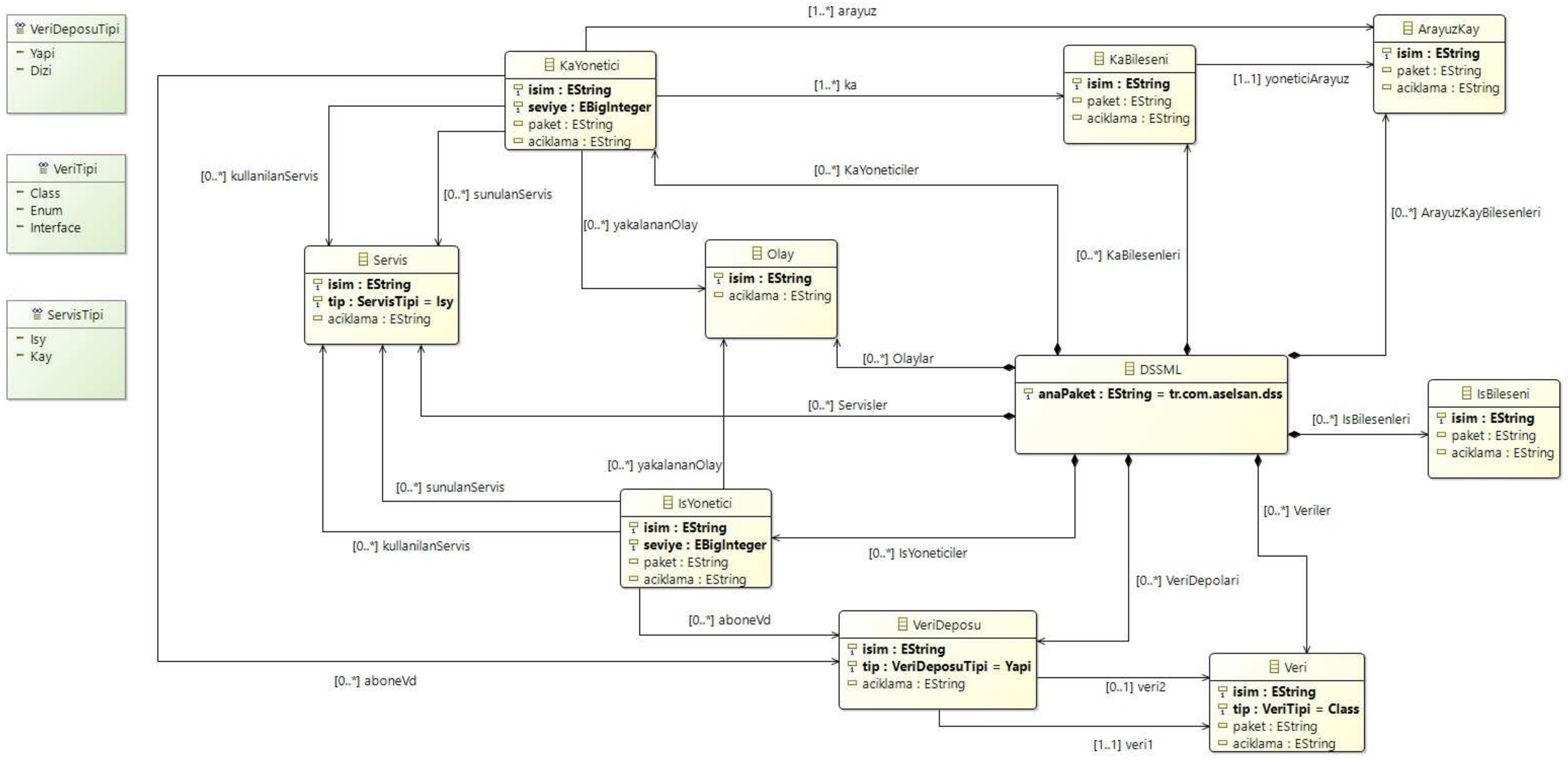


Figure A.5: DSS-RSA Domain Model

APPENDIX B

GMF TOOLING WORKFLOW

B.1 Creating GMF Project

We create a new project to hold our models by clicking on the File menu of Eclipse and selecting "$New \rightarrow Other \rightarrow Graphical\ editor\ project$". We give the project the name "com.aselsan.dss.dssml" and click "Show dashboard view for the created project".

GMF has a utility called the GMF dashboard that facilitates the process of generating a graphical modeling editor. Figure B.1 shows the overwiew of GMF Dashboard.

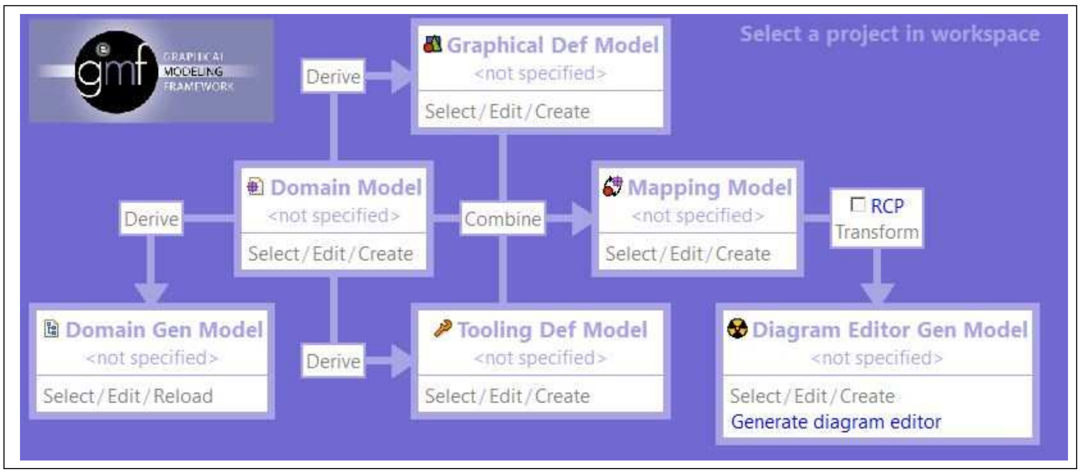


Figure B.1: GMF Dashboard

B.2 Developing Domain Model

DSS-RSA metamodel is copied in model folder of the project and selected by clicking "Select" option of Domain Model on GMF Dashboard.

After Domain Model is selected, we click "Derive" between Domain Model and Domain Gen Model on GMF Dashboard and then, a wizard to create Gen Model is opened. We give the Gen Model the name "metamodel.genmodel" and use the defaults of the wizard. Gen Model creation wizard can also be opened without using GMF Dashboard by right-clicking the metamodel file and selecting "$New \rightarrow Other \rightarrow EMF\ Generator\ Model$".

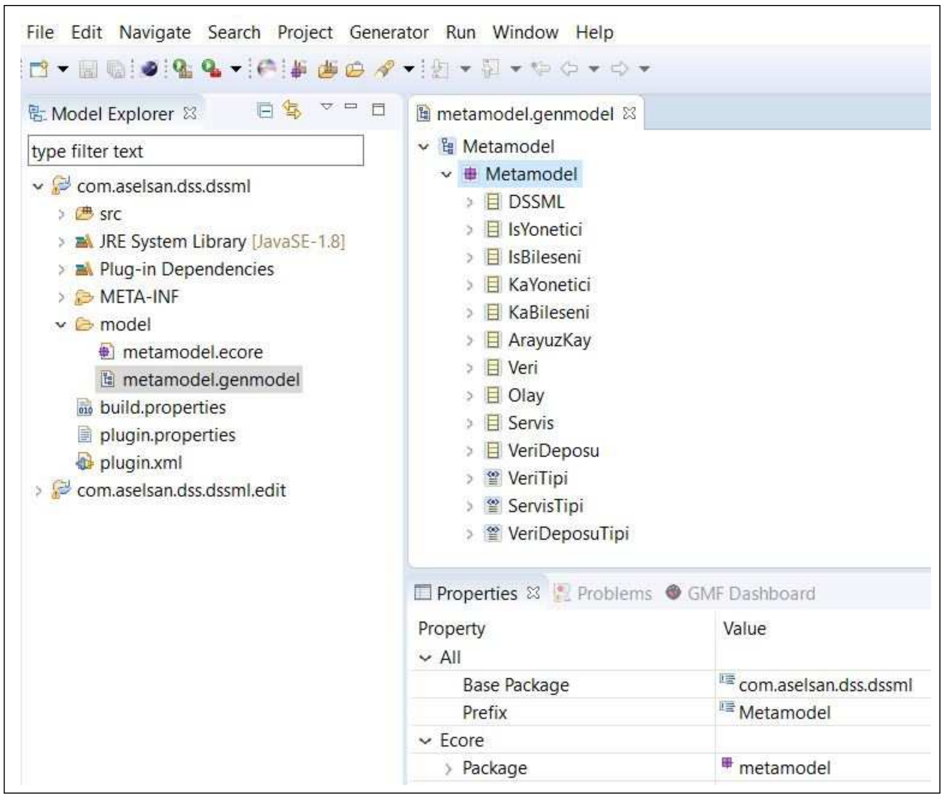
After Gen Model is created, Metamodel package below the root of the Gen Model is selected and by using the properties view, Base Package property is changed to "com.aselsan.dss.dssml". The root of the Gen Model is right-clicked and Generate Model Code followed by Generate Edit Code is selected. The overwiew of the project after Domain Gen Model is created is shown in Figure B.2.

B.3 Developing Graphical Definition Model

To develop Graphical Definition Model, we click "Derive" between Domain Model and Graphical Def Model on GMF Dashboard. Then, a wizard to create Graphical Definition Model is opened. Graphical Definition Model creation wizard can also be opened without using GMF Dashboard by right-clicking the model folder and selecting "$New \rightarrow Other \rightarrow GMFGraph\ Model$". We give the Graphical Definition Model the name "metamodel.gmfgraph" and after loading metamodel we choose "DSSML" diagram element as root element. Domain Model view of Graphical Definition Model creation wizard is shown in Figure B.3.

After clicking Next button in Figure B.3, we specify basic graphical definition of the Domain Model as shown in Figure B.4, Figure B.5 and Figure B.6. After clicking the Finish button in Figure B.6, Graphical Definition Model is created.

To color figures in graphical modeling editor, created Graphical Definition Model is



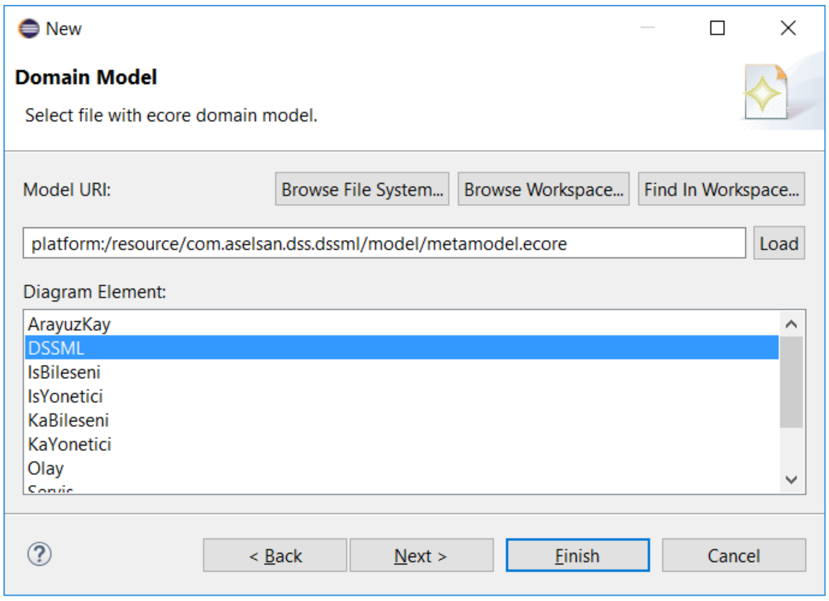
com.aselsan.dss.dssml
- src
- JRE System Library [JavaSE-1.8]
- Plug-in Dependencies
- META-INF
- model
  - metamodel.ecore
  - metamodel.genmodel
- build.properties
- plugin.properties
- plugin.xml

com.aselsan.dss.dssml.edit

Metamodel
- Metamodel
  - DSSML
  - IsYonetici
  - IsBileseni
  - KaYonetici
  - KaBileseni
  - ArayuzKay
  - Veri
  - Olay
  - Servis
  - VeriDeposu
  - VeriTipi
  - ServisTipi
  - VeriDeposuTipi

Property	Value
All	
Base Package	com.aselsan.dss.dssml
Prefix	Metamodel
Ecore	
Package	metamodel

Figure B.2: Overview of the project after Domain Gen Model is created



Domain Model

Select file with ecore domain model.

Model URI:

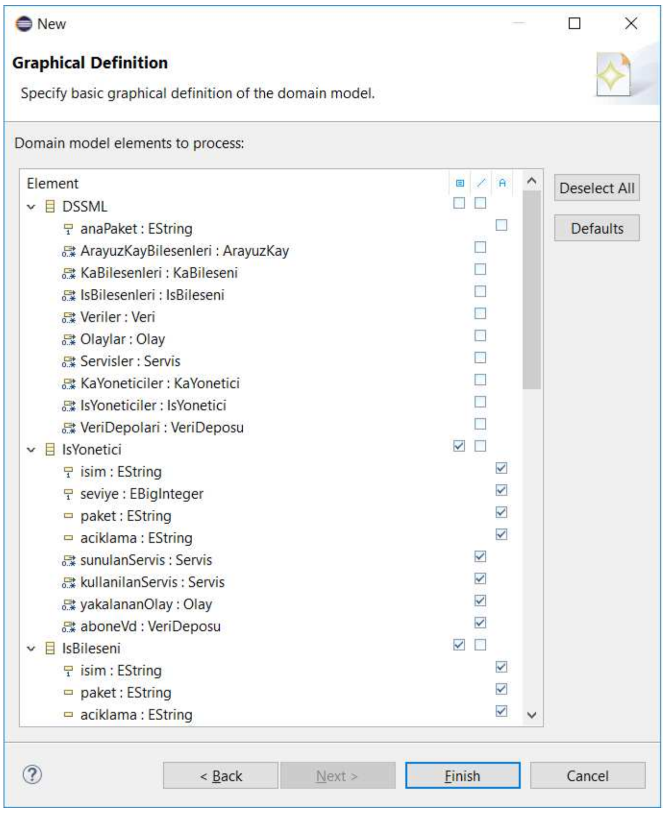
platform:/resource/com.aselsan.dss.dssml/model/metamodel.ecore

Diagram Element:

ArayuzKay
DSSML
IsBileseni
IsYonetici
KaBileseni
KaYonetici
Olay

Figure B.3: Graphical Definition Model creation wizard: select Domain Model

opened and under each Figure Descriptor's Rectangle element, Background Color RGB Color is added as a child element. By using properties view blue, green and red



New

Graphical Definition

Specify basic graphical definition of the domain model.

Domain model elements to process:

Element			
DSSML	☐	☐	
anaPaket : EString			☐
ArayuzKayBilesenleri : ArayuzKay		☐	
KaBilesenleri : KaBileseni		☐	
IsBilesenleri : IsBileseni		☐	
Veriler : Veri		☐	
Olaylar : Olay		☐	
Servisler : Servis		☐	
KaYoneticiler : KaYonetici		☐	
IsYoneticiler : IsYonetici		☐	
VeriDepolari : VeriDeposu		☐	
IsYonetici	☑	☐	
isim : EString			☑
seviye : EBigInteger			☑
paket : EString			☑
aciklama : EString			☑
sunulanServis : Servis		☑	
kullanilanServis : Servis		☑	
yakalananOlay : Olay		☑	
aboneVd : VeriDeposu		☑	
IsBileseni	☑	☐	
isim : EString			☑
paket : EString			☑
aciklama : EString			☑

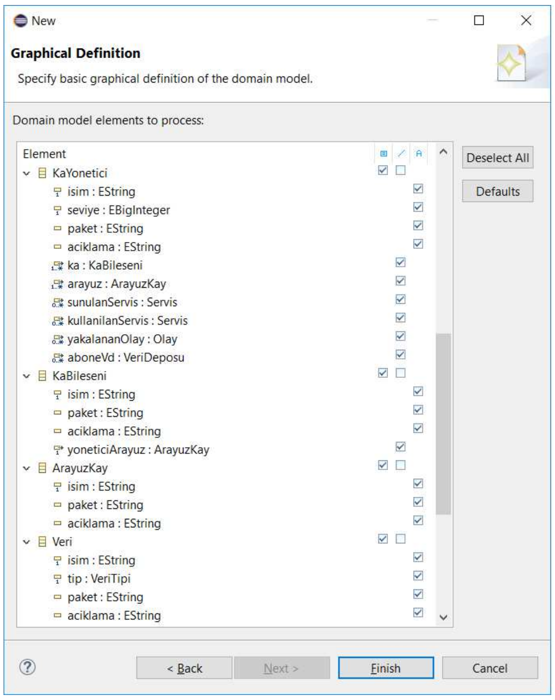
Figure B.4: Graphical Definition Model creation wizard: specify basic graphical definition of the Domain Model Part 1

values of Background Color RGB Color are set as shown in Figure B.7.

B.4 Developing Tooling Definition Model

To develop Tooling Definition Model, we click “Derive” between Domain Model and Tooling Def Model on GMF Dashboard. Then, a wizard to create Tooling Definition Model is opened. Tooling Definition Model creation wizard can also be opened without using GMF Dashboard by right-clicking the model folder and selecting “$New \rightarrow Other \rightarrow GMFTool\ Model$”. We give the Tooling Definition Model the name “metamodel.gmftool” and after loading metamodel we choose “DSSML” diagram element as root element. Domain Model view of Tooling Definition Model creation wizard is shown in Figure B.8.

After clicking Next button in Figure B.8, we specify basic tooling definition of the


New

Graphical Definition

Specify basic graphical definition of the domain model.

Domain model elements to process:

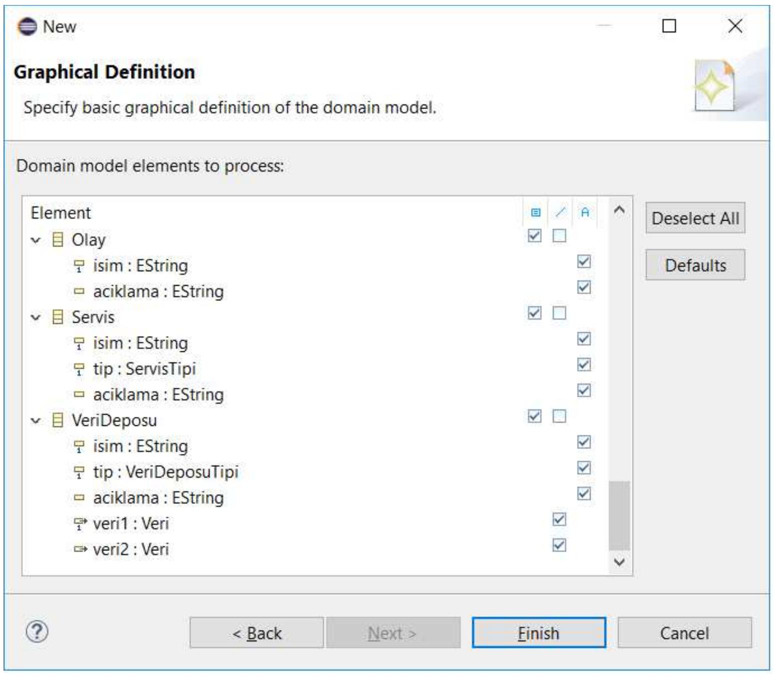
Element	▤	⁄	A
KaYonetici	☑	☐	
isim : EString			☑
seviye : EBigInteger			☑
paket : EString			☑
aciklama : EString			☑
ka : KaBileseni		☑	
arayuz : ArayuzKay		☑	
sunulanServis : Servis		☑	
kullanilanServis : Servis		☑	
yakalananOlay : Olay		☑	
aboneVd : VeriDeposu		☑	
KaBileseni	☑	☐	
isim : EString			☑
paket : EString			☑
aciklama : EString			☑
yoneticiArayuz : ArayuzKay		☑	
ArayuzKay	☑	☐	
isim : EString			☑
paket : EString			☑
aciklama : EString			☑
Veri	☑	☐	
isim : EString			☑
tip : VeriTipi			☑
paket : EString			☑
aciklama : EString			☑

Deselect All

Defaults

< Back   Next >   Finish   Cancel

Figure B.5: Graphical Definition Model creation wizard: specify basic graphical definition of the Domain Model Part 2


New

Graphical Definition

Specify basic graphical definition of the domain model.

Domain model elements to process:

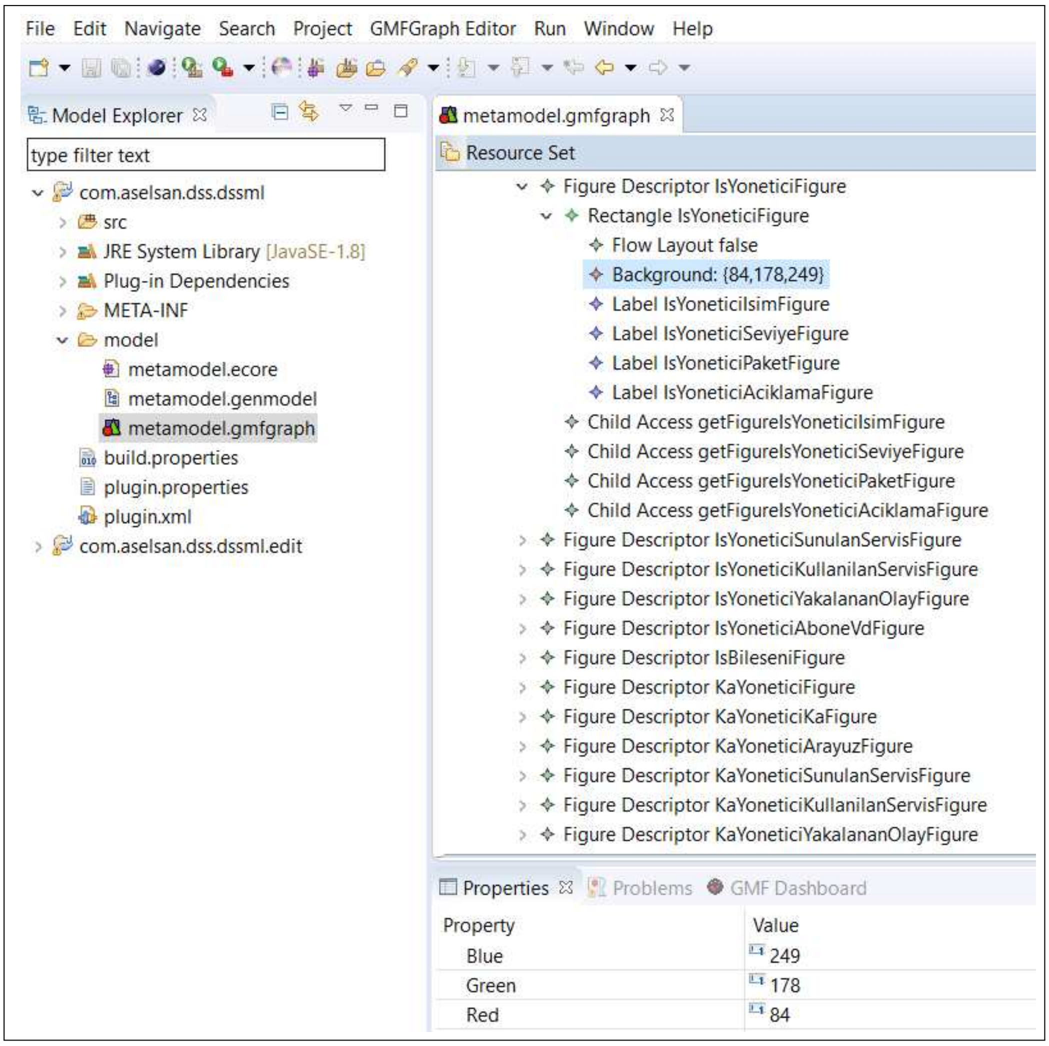
Element	▤	⁄	A
Olay	☑	☐	
isim : EString			☑
aciklama : EString			☑
Servis	☑	☐	
isim : EString			☑
tip : ServisTipi			☑
aciklama : EString			☑
VeriDeposu	☑	☐	
isim : EString			☑
tip : VeriDeposuTipi			☑
aciklama : EString			☑
veri1 : Veri		☑	
veri2 : Veri		☑	

Deselect All

Defaults

< Back   Next >   Finish   Cancel

Figure B.6: Graphical Definition Model creation wizard: specify basic graphical definition of the Domain Model Part 3



com.aselsan.dss.dssml
- src
- JRE System Library [JavaSE-1.8]
- Plug-in Dependencies
- META-INF
- model
  - metamodel.ecore
  - metamodel.genmodel
  - metamodel.gmfgraph
- build.properties
- plugin.properties
- plugin.xml

com.aselsan.dss.dssml.edit

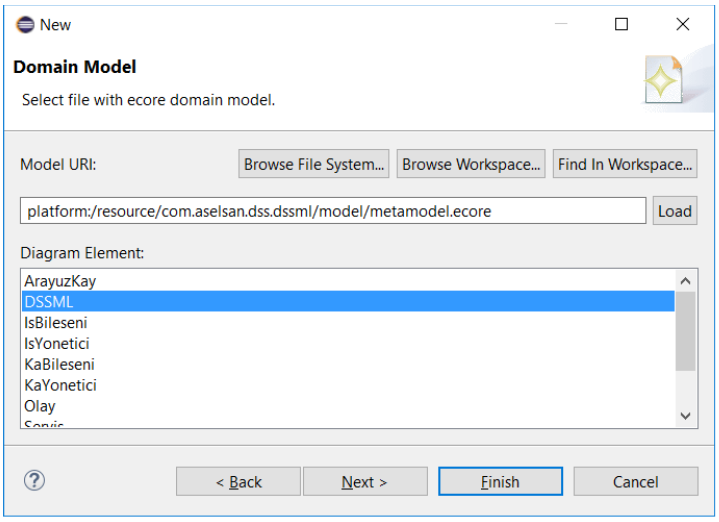
Resource Set
- Figure Descriptor IsYoneticiFigure
  - Rectangle IsYoneticiFigure
    - Flow Layout false
    - Background: {84,178,249}
    - Label IsYoneticiIsimFigure
    - Label IsYoneticiSeviyeFigure
    - Label IsYoneticiPaketFigure
    - Label IsYoneticiAciklamaFigure
  - Child Access getFigureIsYoneticiIsimFigure
  - Child Access getFigureIsYoneticiSeviyeFigure
  - Child Access getFigureIsYoneticiPaketFigure
  - Child Access getFigureIsYoneticiAciklamaFigure
- Figure Descriptor IsYoneticiSunulanServisFigure
- Figure Descriptor IsYoneticiKullanilanServisFigure
- Figure Descriptor IsYoneticiYakalananOlayFigure
- Figure Descriptor IsYoneticiAboneVdFigure
- Figure Descriptor IsBileseniFigure
- Figure Descriptor KaYoneticiFigure
- Figure Descriptor KaYoneticiKaFigure
- Figure Descriptor KaYoneticiArayuzFigure
- Figure Descriptor KaYoneticiSunulanServisFigure
- Figure Descriptor KaYoneticiKullanilanServisFigure
- Figure Descriptor KaYoneticiYakalananOlayFigure

Property	Value
Blue	249
Green	178
Red	84

Figure B.7: Setting RGB colors of Figures in Graphical Definition Model by using properties view

Domain Model as shown in Figure B.9. After clicking the Finish button in Figure B.9, Tooling Definition Model is created.

To edit Tool Groups, created Tooling Definition Model is opened and existing Tool Group title is updated as “DSS RSA Elemanları” by using properties view. A new Tool Group is added and title of this Tool Group is set as “Bağlantılar”. Then, Creation Tools that correspond to the relations of DSS RSA are moved under “Bağlantılar” Tool Group. Overview of the Tooling Definition Model after Tool Groups are updated is shown in Figure B.10.


New

Domain Model

Select file with ecore domain model.

Model URI: Browse File System... Browse Workspace... Find In Workspace...

platform:/resource/com.aselsan.dss.dssml/model/metamodel.ecore

Diagram Element:

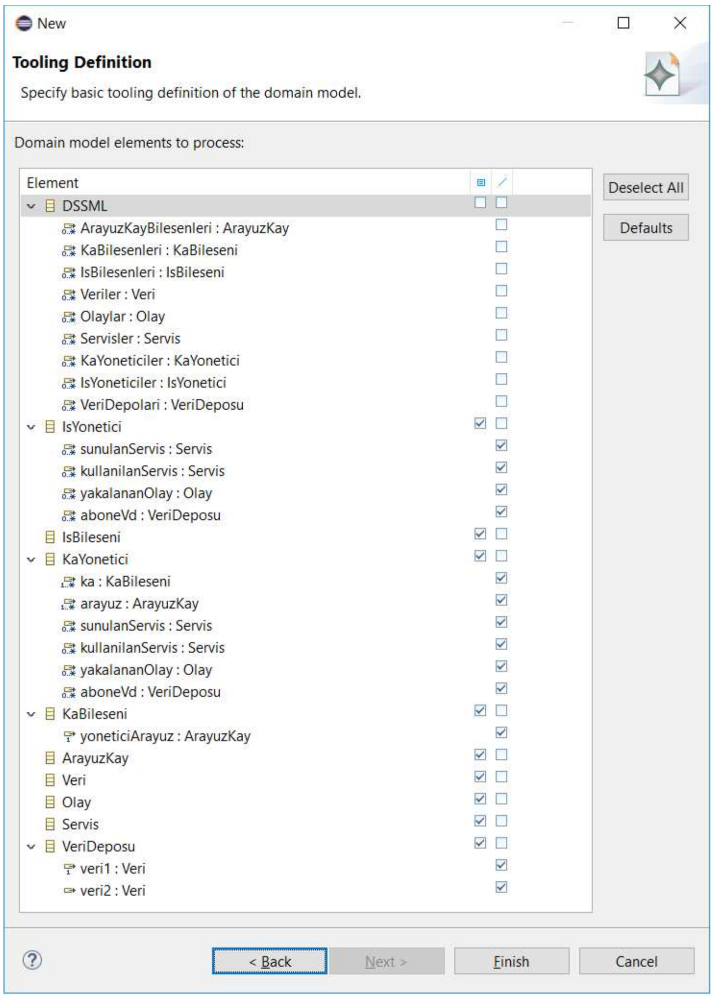
ArayuzKay
DSSML
IsBileseni
IsYonetici
KaBileseni
KaYonetici
Olay

Figure B.8: Tooling Definition Model creation wizard: select Domain Model

B.5 Developing Mapping Model

To develop Mapping Model, we click "Combine" between Domain Model, Graphical Def Model, Tooling Def Model and Mapping Model on GMF Dashboard. Then, a wizard to create Mapping Model is opened. Mapping Model creation wizard can also be opened without using GMF Dashboard by right-clicking the model folder and selecting "$New \rightarrow Other \rightarrow GMFMap\ Model$". We give the Mapping Model the name "metamodel.gmfmap" and after loading Domain Model we choose "DSSML" diagram element for canvas mapping. Select Domain Model view of Mapping Model creation wizard is shown in Figure B.11.

After clicking the next button in Figure B.11, we load Tooling Definition Model and select diagram palette for canvas mapping as shown in Figure B.12. After clicking the next button in Figure B.12, we load Graphical Definition Model and select diagram canvas for canvas mapping as shown in Figure B.13. After clicking the next button in Figure B.13, we map Domain Model elements as Nodes and Links as shown in Figure B.14. After clicking the Finish button in Figure B.14, Mapping Model is created. Node and Link Mappings of Mapping Model is shown in Figure B.15. Diagram Nodes and Tools can be updated by using properties view.


New

Tooling Definition

Specify basic tooling definition of the domain model.

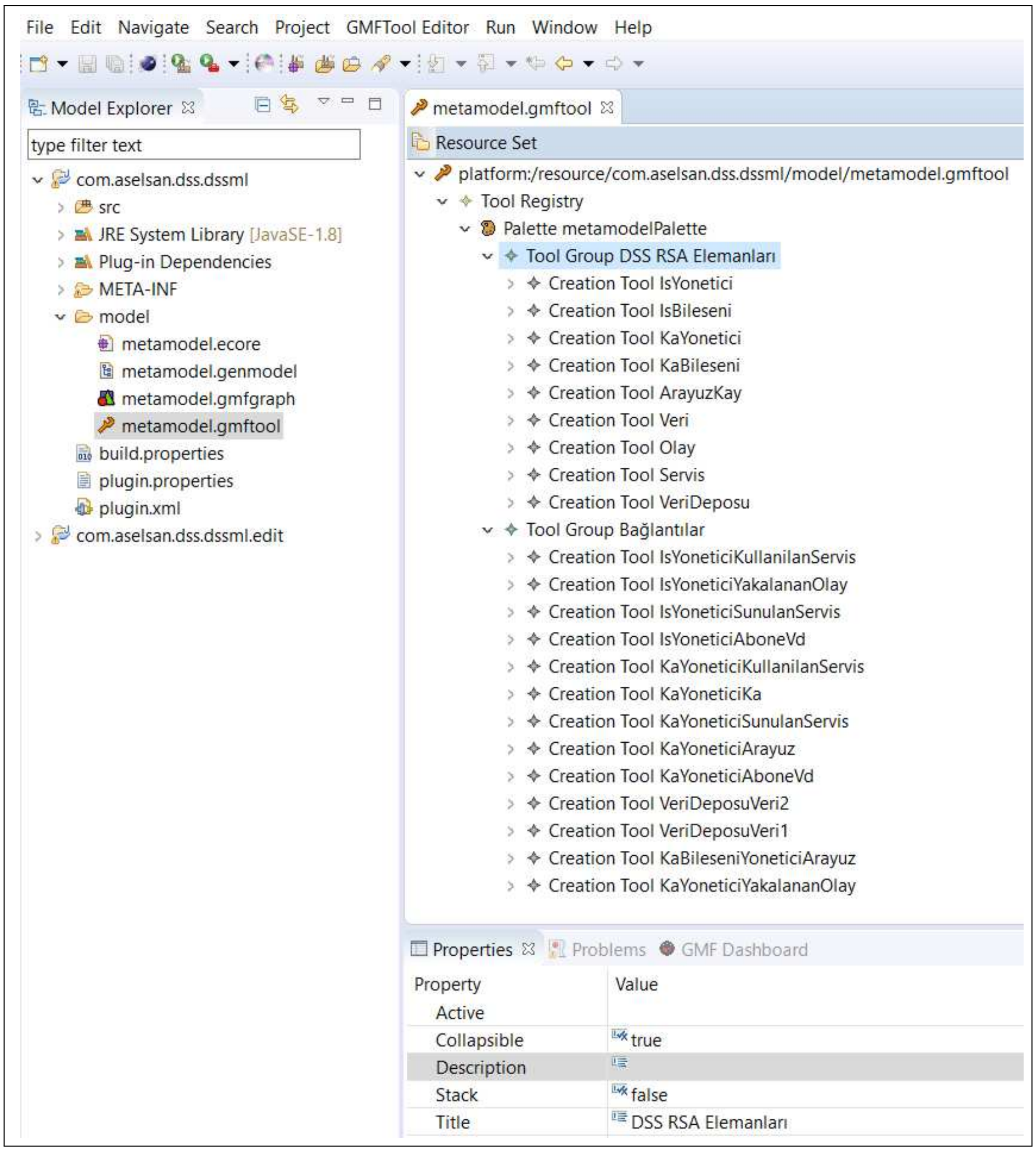
Domain model elements to process:

Element		
DSSML	☐	☐
ArayuzKayBilesenleri : ArayuzKay		☐
KaBilesenleri : KaBileseni		☐
IsBilesenleri : IsBileseni		☐
Veriler : Veri		☐
Olaylar : Olay		☐
Servisler : Servis		☐
KaYoneticiler : KaYonetici		☐
IsYoneticiler : IsYonetici		☐
VeriDepolari : VeriDeposu		☐
IsYonetici	☑	☐
sunulanServis : Servis		☑
kullanilanServis : Servis		☑
yakalananOlay : Olay		☑
aboneVd : VeriDeposu		☑
IsBileseni	☑	☐
KaYonetici	☑	☐
ka : KaBileseni		☑
arayuz : ArayuzKay		☑
sunulanServis : Servis		☑
kullanilanServis : Servis		☑
yakalananOlay : Olay		☑
aboneVd : VeriDeposu		☑
KaBileseni	☑	☐
yoneticiArayuz : ArayuzKay		☑
ArayuzKay	☑	☐
Veri	☑	☐
Olay	☑	☐
Servis	☑	☐
VeriDeposu	☑	☐
veri1 : Veri		☑
veri2 : Veri		☑

Figure B.9: Tooling Definition Model creation wizard: specify basic tooling definition of the Domain Model

B.6 Creating Generator Model and Diagram Plug-in

To create Generator Model, we click "Transform" between Mapping Model and Diagram Editor Gen Model on GMF Dashboard. Then, Generator Model is created easily under model folder with the name "metamodel.gmfgen". Generator Model can also be created without using GMF Dashboard by right-clicking Mapping Model and selecting Create Generator Model. After giving the name "metamodel.gmfgen" to Generator Model, Mapping Model is loaded as shown in Figure B.16. After clicking the Next button in Figure B.16, GenModel is loaded as shown in Figure B.17. After clicking the Next button in Figure B.17, transformation options are specified as shown



```
com.aselsan.dss.dssml
  src
  JRE System Library [JavaSE-1.8]
  Plug-in Dependencies
  META-INF
  model
    metamodel.ecore
    metamodel.genmodel
    metamodel.gmfgraph
    metamodel.gmftool
  build.properties
  plugin.properties
  plugin.xml
com.aselsan.dss.dssml.edit
```

```
platform:/resource/com.aselsan.dss.dssml/model/metamodel.gmftool
  Tool Registry
    Palette metamodelPalette
      Tool Group DSS RSA Elemanları
        Creation Tool IsYonetici
        Creation Tool IsBileseni
        Creation Tool KaYonetici
        Creation Tool KaBileseni
        Creation Tool ArayuzKay
        Creation Tool Veri
        Creation Tool Olay
        Creation Tool Servis
        Creation Tool VeriDeposu
      Tool Group Bağlantılar
        Creation Tool IsYoneticiKullanilanServis
        Creation Tool IsYoneticiYakalananOlay
        Creation Tool IsYoneticiSunulanServis
        Creation Tool IsYoneticiAboneVd
        Creation Tool KaYoneticiKullanilanServis
        Creation Tool KaYoneticiKa
        Creation Tool KaYoneticiSunulanServis
        Creation Tool KaYoneticiArayuz
        Creation Tool KaYoneticiAboneVd
        Creation Tool VeriDeposuVeri2
        Creation Tool VeriDeposuVeri1
        Creation Tool KaBileseniYoneticiArayuz
        Creation Tool KaYoneticiYakalananOlay
```

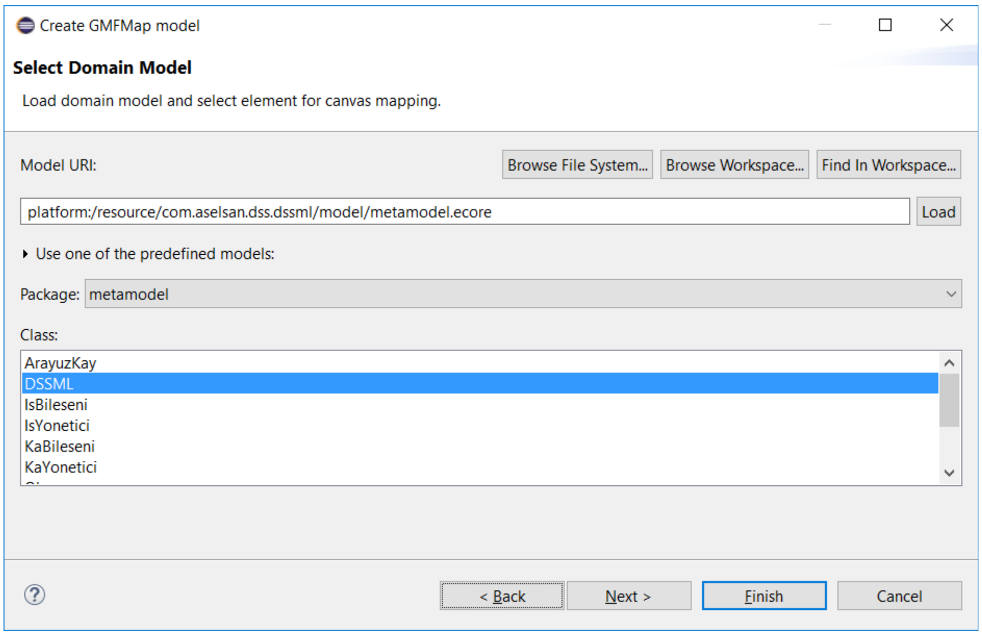
Property	Value
Active	
Collapsible	true
Description	
Stack	false
Title	DSS RSA Elemanları

Figure B.10: Updating Tool Groups in Tooling Definition Model by using properties view

in Figure B.18. After clicking the Finish button in Figure B.18, Generator Model is created.

To use defined OCL constraints in Ecore Model in order to validate the created model, the Generator Model is opened and “Gen Diagram DSSMLEditPart” is clicked. By using the properties view, “Validation Decorators” and “Validation Enabled” values are changed to “true” as shown in Figure B.19.

To generate diagram code, Generator Model is right-clicked and Generate Diagram


Create GMFMap model

Select Domain Model

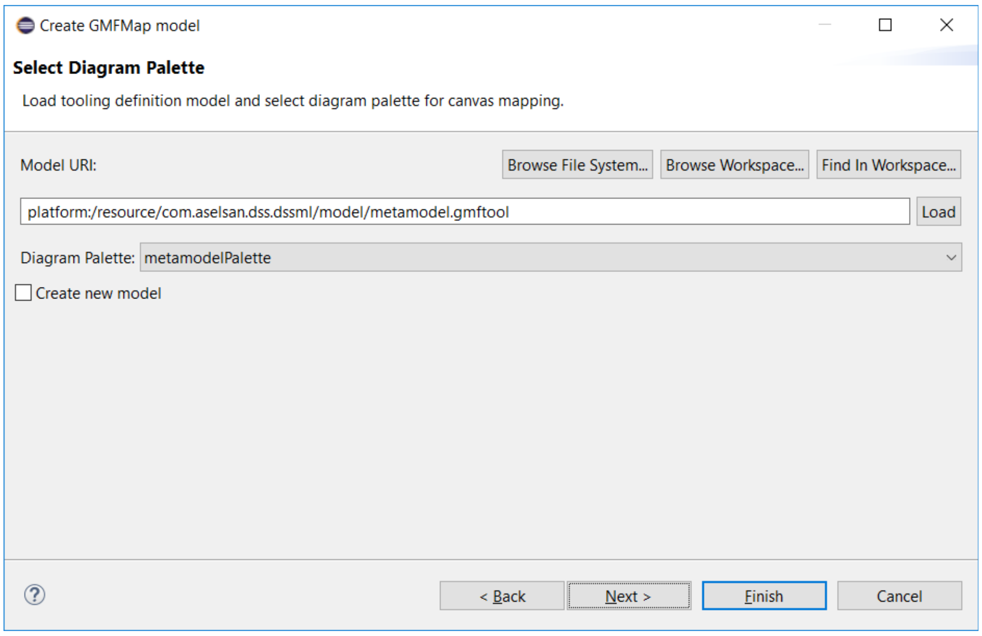
Load domain model and select element for canvas mapping.

Model URI: platform:/resource/com.aselsan.dss.dssml/model/metamodel.ecore

Package: metamodel

Class:
- ArayuzKay
- DSSML
- IsBileseni
- IsYonetici
- KaBileseni
- KaYonetici

Figure B.11: Mapping Model creation wizard: load Domain Model and select element for canvas mapping


Create GMFMap model

Select Diagram Palette

Load tooling definition model and select diagram palette for canvas mapping.

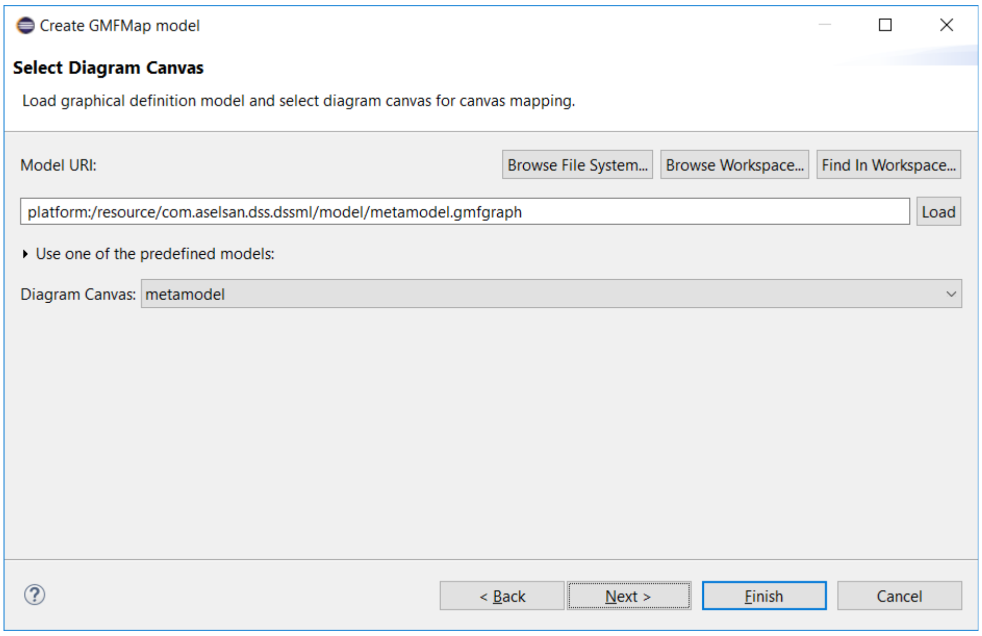
Model URI: platform:/resource/com.aselsan.dss.dssml/model/metamodel.gmftool

Diagram Palette: metamodelPalette

[ ] Create new model

Figure B.12: Mapping Model creation wizard: load Tooling Definition Model and select diagram palette for canvas mapping

Code is selected. Generated diagram project is right-clicked and run as an Eclipse Application. After the application is launched, a new Java Project is created and a


Create GMFMap model

Select Diagram Canvas

Load graphical definition model and select diagram canvas for canvas mapping.

Model URI: Browse File System... Browse Workspace... Find In Workspace...

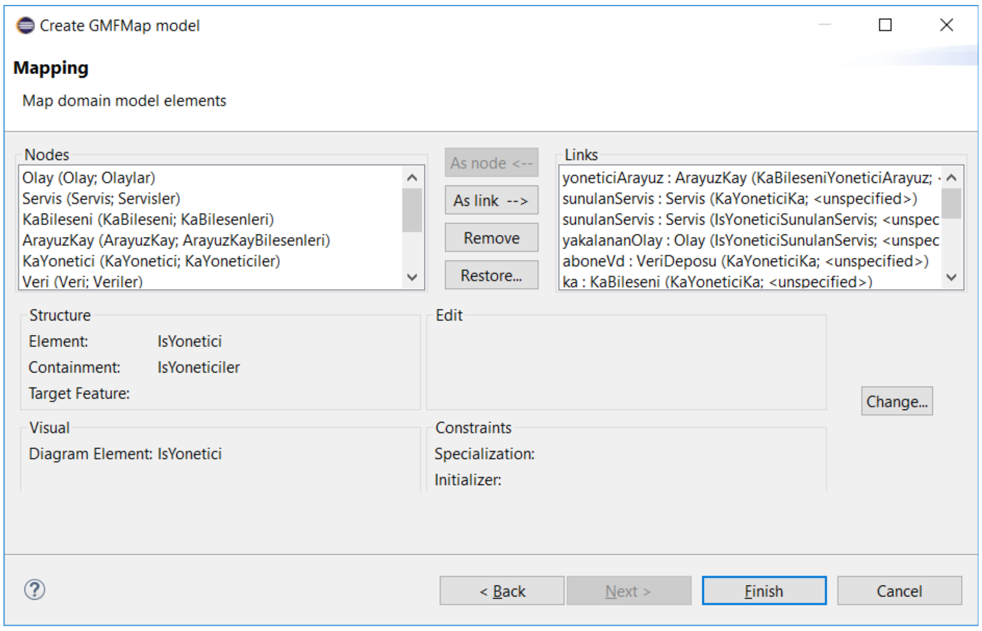
platform:/resource/com.aselsan.dss.dssml/model/metamodel.gmfgraph Load

Use one of the predefined models:

Diagram Canvas: metamodel

< Back Next > Finish Cancel

Figure B.13: Mapping Model creation wizard: load Graphical Definition Model and select diagram canvas for canvas mapping


Create GMFMap model

Mapping

Map domain model elements

Nodes
Olay (Olay; Olaylar)
Servis (Servis; Servisler)
KaBileseni (KaBileseni; KaBilesenleri)
ArayuzKay (ArayuzKay; ArayuzKayBilesenleri)
KaYonetici (KaYonetici; KaYoneticiler)
Veri (Veri; Veriler)

As node <-- As link --> Remove Restore...

Links
yoneticiArayuz : ArayuzKay (KaBileseniYoneticiArayuz;
sunulanServis : Servis (KaYoneticiKa; <unspecified>)
sunulanServis : Servis (IsYoneticiSunulanServis; <unspec
yakalananOlay : Olay (IsYoneticiSunulanServis; <unspec
aboneVd : VeriDeposu (KaYoneticiKa; <unspecified>)
ka : KaBileseni (KaYoneticiKa; <unspecified>)

Structure
Element: IsYonetici
Containment: IsYoneticiler
Target Feature:

Edit

Change...

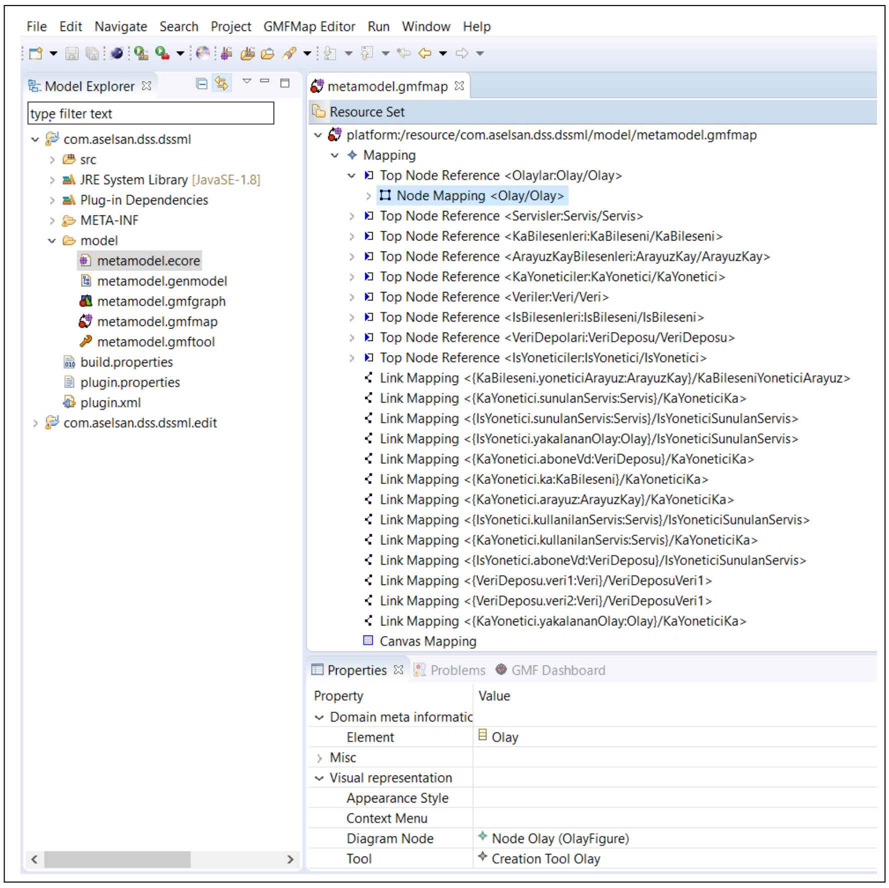
Visual
Diagram Element: IsYonetici

Constraints
Specialization:
Initializer:

< Back Next > Finish Cancel

Figure B.14: Mapping Model creation wizard: map Domain Model elements

new "Metamodel Diagram" is added to the Java Project. GMF Diagram Editor is ready now to create a model as shown in Figure B.20.

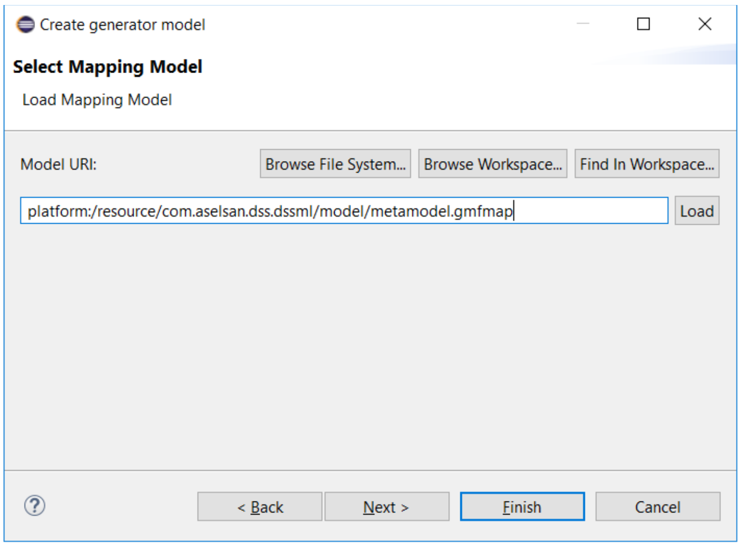


```
com.aselsan.dss.dssml
  src
  JRE System Library [JavaSE-1.8]
  Plug-in Dependencies
  META-INF
  model
    metamodel.ecore
    metamodel.genmodel
    metamodel.gmfgraph
    metamodel.gmfmap
    metamodel.gmftool
  build.properties
  plugin.properties
  plugin.xml
com.aselsan.dss.dssml.edit
```

```
platform:/resource/com.aselsan.dss.dssml/model/metamodel.gmfmap
  Mapping
    Top Node Reference <Olaylar:Olay/Olay>
      Node Mapping <Olay/Olay>
    Top Node Reference <Servisler:Servis/Servis>
    Top Node Reference <KaBilesenleri:KaBileseni/KaBileseni>
    Top Node Reference <ArayuzKayBilesenleri:ArayuzKay/ArayuzKay>
    Top Node Reference <KaYoneticiler:KaYonetici/KaYonetici>
    Top Node Reference <Veriler:Veri/Veri>
    Top Node Reference <IsBilesenleri:IsBileseni/IsBileseni>
    Top Node Reference <VeriDepolari:VeriDeposu/VeriDeposu>
    Top Node Reference <IsYoneticiler:IsYonetici/IsYonetici>
    Link Mapping <{KaBileseni.yoneticiArayuz:ArayuzKay}/KaBileseniYoneticiArayuz>
    Link Mapping <{KaYonetici.sunulanServis:Servis}/KaYoneticiKa>
    Link Mapping <{IsYonetici.sunulanServis:Servis}/IsYoneticiSunulanServis>
    Link Mapping <{IsYonetici.yakalananOlay:Olay}/IsYoneticiSunulanServis>
    Link Mapping <{KaYonetici.aboneVd:VeriDeposu}/KaYoneticiKa>
    Link Mapping <{KaYonetici.ka:KaBileseni}/KaYoneticiKa>
    Link Mapping <{KaYonetici.arayuz:ArayuzKay}/KaYoneticiKa>
    Link Mapping <{IsYonetici.kullanilanServis:Servis}/IsYoneticiSunulanServis>
    Link Mapping <{KaYonetici.kullanilanServis:Servis}/KaYoneticiKa>
    Link Mapping <{IsYonetici.aboneVd:VeriDeposu}/IsYoneticiSunulanServis>
    Link Mapping <{VeriDeposu.veri1:Veri}/VeriDeposuVeri1>
    Link Mapping <{VeriDeposu.veri2:Veri}/VeriDeposuVeri1>
    Link Mapping <{KaYonetici.yakalananOlay:Olay}/KaYoneticiKa>
    Canvas Mapping
```

Property	Value
Domain meta informatic	
Element	Olay
Misc	
Visual representation	
Appearance Style	
Context Menu	
Diagram Node	Node Olay (OlayFigure)
Tool	Creation Tool Olay

Figure B.15: Mapping Model

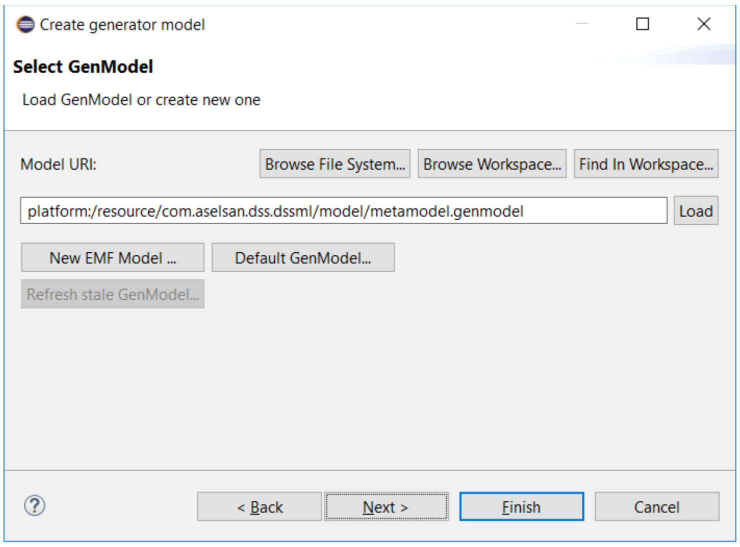


Select Mapping Model

Load Mapping Model

Model URI: platform:/resource/com.aselsan.dss.dssml/model/metamodel.gmfmap

Figure B.16: Generator Model creation wizard: load Mapping Model

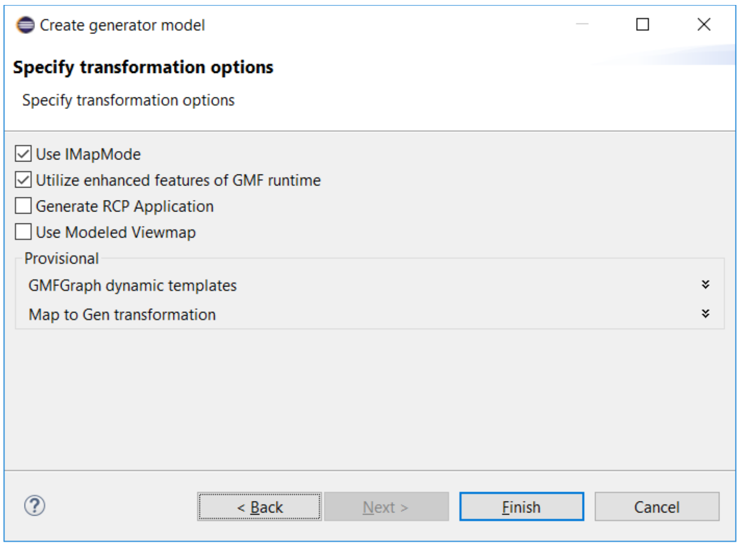

Create generator model

Select GenModel

Load GenModel or create new one

Model URI: platform:/resource/com.aselsan.dss.dssml/model/metamodel.genmodel

Figure B.17: Generator Model creation wizard: load GenModel


Create generator model

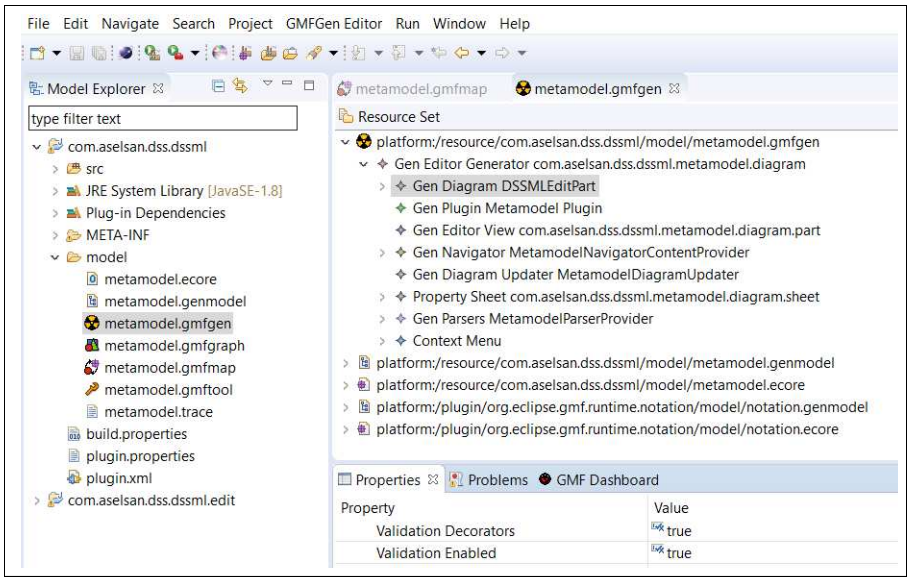
Specify transformation options

Specify transformation options

- [✓] Use IMapMode
- [✓] Utilize enhanced features of GMF runtime
- [ ] Generate RCP Application
- [ ] Use Modeled Viewmap

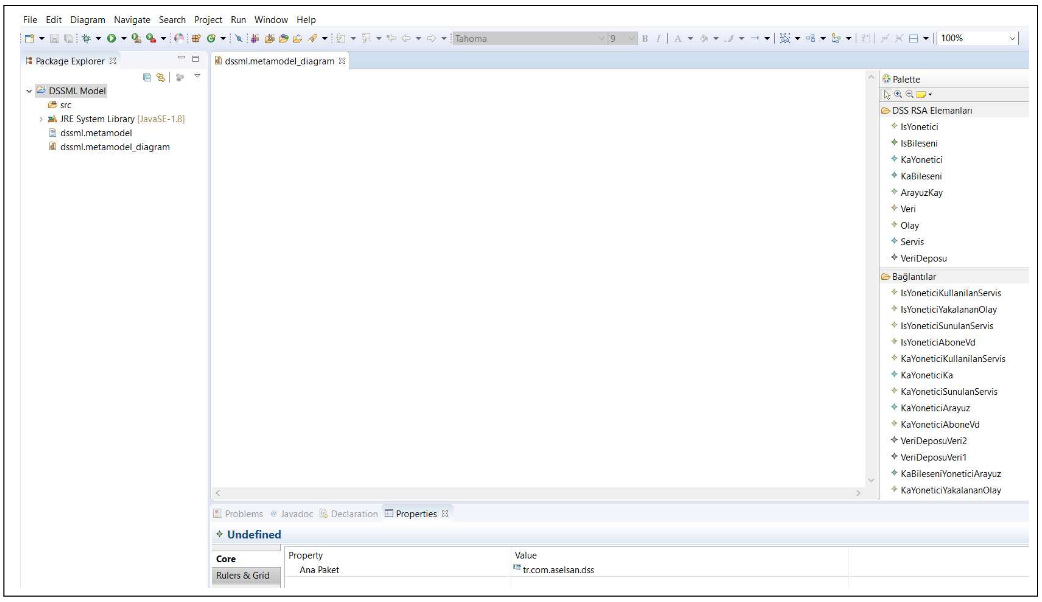
Provisional

Figure B.18: Generator Model creation wizard: specify transformation options


File Edit Navigate Search Project GMFGen Editor Run Window Help
Model Explorer
type filter text
com.aselsan.dss.dssml
src
JRE System Library [JavaSE-1.8]
Plug-in Dependencies
META-INF
model
metamodel.ecore
metamodel.genmodel
metamodel.gmfgen
metamodel.gmfgraph
metamodel.gmfmap
metamodel.gmftool
metamodel.trace
build.properties
plugin.properties
plugin.xml
com.aselsan.dss.dssml.edit
metamodel.gmfmap
metamodel.gmfgen
Resource Set
platform:/resource/com.aselsan.dss.dssml/model/metamodel.gmfgen
Gen Editor Generator com.aselsan.dss.dssml.metamodel.diagram
Gen Diagram DSSMLEditPart
Gen Plugin Metamodel Plugin
Gen Editor View com.aselsan.dss.dssml.metamodel.diagram.part
Gen Navigator MetamodelNavigatorContentProvider
Gen Diagram Updater MetamodelDiagramUpdater
Property Sheet com.aselsan.dss.dssml.metamodel.diagram.sheet
Gen Parsers MetamodelParserProvider
Context Menu
platform:/resource/com.aselsan.dss.dssml/model/metamodel.genmodel
platform:/resource/com.aselsan.dss.dssml/model/metamodel.ecore
platform:/plugin/org.eclipse.gmf.runtime.notation/model/notation.genmodel
platform:/plugin/org.eclipse.gmf.runtime.notation/model/notation.ecore
Properties
Problems
GMF Dashboard

Property	Value
Validation Decorators	true
Validation Enabled	true

Figure B.19: Activating OCL constraints on Generator Model



Property	Value
Ana Paket	tr.com.aselsan.dss

Figure B.20: GMF Diagram Editor