Component Reusability Analysis for Exchanging Electronic Health Records

Jaechang Nam
This thesis is submitted to the Department of Interaction and System Design, School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

University advisor(s):
Shahid Hussain, Prof. Rune Gustavsson
School of Computing

School of Computing
Blekinge Institute of Technology
Soft Center
SE–37225 Ronneby
Sweden
ABSTRACT

As Information Communication Technologies (ICTs) are growing, there have been ceaseless efforts to develop a National Health Information Infrastructure (NHII). One of the challenges in constructing a NHII is concerned with the management of Electronic Health Records (EHRs). In particular, exchanging EHRs is an important factor in establishing interoperability within a NHII, and the reusability of the functionality for exchanging EHRs is one of major solutions to construct an NHII. In this study, we obtain several component models, and conduct empirical studies to validate the component models in terms of component reusability.

Using HL7 CDA (Health Level 7 Clinical Document Architecture) as an EHR standard, we implemented three prototypes of the EHR Exchanger based on JavaBeans, the exogenous connectors and the mediator connector respectively. As shown in the experiment results, the reuse approach using a mediator connector leads to better component reusability in terms of external dependency, total coupling between objects (CBO), additional lines of codes (LOC), and performance. Thus, we believe that the reuse approach using a mediator connector yields many benefits in terms of component reusability for the EHR Exchanger implementation.

Keywords: Electronic Health Records (EHRs), component reuse approach, software component model, mediator connector
ACKNOWLEDGEMENTS

I dedicate this thesis to the Heavenly Father and my lovely wife. The supports from my parents and two elder sisters are always more than enough as well. I appreciate their help. And I couldn’t forget my friends at RCCG in Karlskrona, particularly all choir members. Without their encouragement and prayer, this work would not be achieved. Finally, I thank for Dr. Gouhua Bai, and Mr. Shahid Hussain as a thesis examiner and a supervisor respectively. They always helped me to finish this work with advices academically and mentally.
TABLE OF CONTENTS

ABSTRACT .............................................................................................................. 2

ACKNOWLEDGEMENTS ...................................................................................... 3

LIST OF ACRONYMS .......................................................................................... 6

INTRODUCTION .................................................................................................... 7

CHAPTER 1: BACKGROUND ............................................................................... 8
  1.1 COMPONENT-BASED DEVELOPMENT ......................................................... 8
  1.2 ELECTRONIC HEALTHCARE ...................................................................... 9
      1.2.1 National healthcare infrastructure ................................................... 9
      1.2.2 Electronic health records ................................................................. 10
  1.3 MOTIVATION ............................................................................................... 10

CHAPTER 2: PROBLEM DEFINITION AND GOALS ............................................. 11
  2.1 PROBLEM DEFINITION ............................................................................. 11
  2.2 GOAL AND OBJECTIVES .......................................................................... 11
  2.3 RESEARCH QUESTIONS ............................................................................ 12
  2.4 EXPECTED OUTCOMES .......................................................................... 13

CHAPTER 3: METHOD .......................................................................................... 15
  3.1 OVERVIEW ................................................................................................. 15
  3.2 LITERATURE REVIEW .............................................................................. 16
      3.2.1 Standards for EHRs and implementation technologies .................... 16
      3.2.2 Overview of software component models ...................................... 19
      3.2.3 Overview of software component evaluation methods ..................... 21
  3.3 PROTOTYPING .......................................................................................... 22
  3.4 EXPERIMENT ............................................................................................ 22

CHAPTER 4: THEORETICAL WORK .................................................................. 24
  4.1 THE CHOSEN EHR STANDARD AND ITS IMPLEMENTATION TECHNOLOGIES 24
      4.1.1 General framework of HL7 CDA ....................................................... 24
      4.1.2 Creation of a HL7 CDA document .................................................... 26
  4.2 UNDERLYING THEORIES OF RECENTLY PROPOSED COMPONENT MODELS 28
      4.2.1 Exogenous connectors ...................................................................... 29
      4.2.2 Mediator connector .......................................................................... 31
      4.2.3 Active Binding Technology ............................................................... 35

CHAPTER 5: EMPIRICAL STUDY ...................................................................... 39
  5.1 PROTOTYPING .......................................................................................... 39
      5.1.1 Basic framework of exchanging EHRs .............................................. 39
      5.1.2 Scenario for the prototype implementation .................................... 39
      5.1.3 Prototype 1: Basic implementation using JavaBeans ....................... 40
      5.1.4 Prototype 2: Implementation using exogenous connectors ............. 41
      5.1.5 Prototype 3: Implementation using mediator connectors ............... 44
  5.2 EVALUATION METHODS FOR COMPONENT AND ITS MODEL ............... 46
  5.3 EXPERIMENT PLAN ................................................................................... 47

CHAPTER 6: RESULTS ....................................................................................... 49
  6.1 MEASUREMENT OF EXTERNAL DEPENDENCY ....................................... 49
  6.2 MEASUREMENT OF TOTAL COUPLING ............................................... 50
  6.3 MEASUREMENT OF ADDITIONAL LINES OF CODE .............................. 51
  6.4 PERFORMANCE MEASUREMENT ............................................................ 52
  6.5 SUMMARY ............................................................................................... 53

CHAPTER 7: DISCUSSION .................................................................................. 54
  7.1 DETAIL INTERPRETATION OF RESULTS ................................................. 54
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1.1</td>
<td>External dependency</td>
<td>54</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Total coupling</td>
<td>54</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Additional lines of code</td>
<td>55</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Performance</td>
<td>55</td>
</tr>
<tr>
<td>7.2</td>
<td>Issues of the Component Model Using a Mediator Connector</td>
<td>56</td>
</tr>
<tr>
<td>7.3</td>
<td>Threat to Validity</td>
<td>56</td>
</tr>
<tr>
<td>7.4</td>
<td>Future Works</td>
<td>57</td>
</tr>
<tr>
<td>7.5</td>
<td>Summary</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>REFERENCES</td>
<td>61</td>
</tr>
</tbody>
</table>
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABT</td>
<td>Active Binding Technology</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institutes</td>
</tr>
<tr>
<td>CBD</td>
<td>Component-Based Development</td>
</tr>
<tr>
<td>CBO</td>
<td>Coupling Between Objects</td>
</tr>
<tr>
<td>CDA</td>
<td>Clinical Document Architecture</td>
</tr>
<tr>
<td>CDL</td>
<td>Component Definition Language</td>
</tr>
<tr>
<td>CEN/TC 251</td>
<td>European Committee for Standardization Technical Committee 251</td>
</tr>
<tr>
<td>CIML</td>
<td>Component Interaction Markup Language</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-The-Shelf</td>
</tr>
<tr>
<td>D-MIM</td>
<td>Domain Message Information Model</td>
</tr>
<tr>
<td>DICOM</td>
<td>Digital Imaging and Communication in Medicine</td>
</tr>
<tr>
<td>EHR</td>
<td>Electronic Health Record</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans</td>
</tr>
<tr>
<td>EMR</td>
<td>Electronic Medical Record</td>
</tr>
<tr>
<td>HIS</td>
<td>Health Information System</td>
</tr>
<tr>
<td>HL7</td>
<td>Health Level 7</td>
</tr>
<tr>
<td>ICTs</td>
<td>Information Communication Technologies</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization of Standardization</td>
</tr>
<tr>
<td>ISO/TC 215</td>
<td>International Organization of Standardization Technical Committee 215</td>
</tr>
<tr>
<td>LLP</td>
<td>Lower Layer Protocol</td>
</tr>
<tr>
<td>LOC</td>
<td>Lines of Code</td>
</tr>
<tr>
<td>LOINC</td>
<td>Logical Observation Identifiers, Names and Codes</td>
</tr>
<tr>
<td>MF</td>
<td>Main Frame</td>
</tr>
<tr>
<td>MLLP</td>
<td>Minimal Lower Layer Protocol</td>
</tr>
<tr>
<td>NEMA</td>
<td>National Electrical Manufacturers Association</td>
</tr>
<tr>
<td>NHII</td>
<td>National Health Information Infrastructure</td>
</tr>
<tr>
<td>NHS</td>
<td>National Health Service</td>
</tr>
<tr>
<td>OC</td>
<td>Originality Components</td>
</tr>
<tr>
<td>PLC</td>
<td>Product Line Engineering</td>
</tr>
<tr>
<td>QC</td>
<td>Quality Components</td>
</tr>
<tr>
<td>R-MIM</td>
<td>Refined Message Information Model</td>
</tr>
<tr>
<td>RC</td>
<td>Reusable Components</td>
</tr>
<tr>
<td>RIM</td>
<td>Reference Information Model</td>
</tr>
<tr>
<td>SCCp</td>
<td>Self-completeness of Component’s parameter</td>
</tr>
<tr>
<td>SCCCr</td>
<td>Self-completeness of Component’s return value</td>
</tr>
<tr>
<td>SNOMED</td>
<td>Systematized Nomenclature of Medicine</td>
</tr>
<tr>
<td>SNOMED CT</td>
<td>SNOMED Clinical Terms</td>
</tr>
<tr>
<td>TDM</td>
<td>Template Definition Model</td>
</tr>
<tr>
<td>TDS</td>
<td>Template Data Schema</td>
</tr>
</tbody>
</table>
INTRODUCTION

After the invention of the Internet in the 20th century, Information Communication Technologies (ICTs) have been applied to all areas from market to government. One of these areas taking advantages from ICTs is health, and health informatics has begun to be on the rise.

In the field of health informatics, the following topics are mainly included: health information system, decision support systems in healthcare, electronic health records, u-health, and international standards for health informatics. Many developed countries are trying to build their National Health Information Infrastructures (NHII) based on these topics [1-3]. NHII is regarded as one solution to improve health care quality, efficiency, and cost issues [1, 3].

In this context, there have been ceaseless efforts to develop NHII. The management of Electronic Health Records (EHRs), which are one of the core components in NHII, is the most important subject. In fact, most research studies regarding NHII mention EHRs as an indispensable component of NHII. EHRs include retrospective, concurrent, and prospective information “to support continuing, efficient and quality integrated health care [4],” classified, according to the International Organization for Standardization (ISO) definition, in several types such as Electronic medical record (EMR) and Personal health record [4]. EHR-related research areas cover standards, the implementation of EHR systems, and interoperability between all healthcare-related systems. Recent research trends in EHRs has been focused on studies of nursing documentation and patient self-documentation, comparison between different information from EHRs for national health record projects around the world and the use of international terminologies for semantic interoperability [4].

Component-based development (CBD) is a leading way to build software systems in software industry and has changed the paradigm of building software “from programming software to compositing software systems [5].” The potential merits of CBD are reduced development time, increased reliability of systems, and increased flexibility [5]. In this regard, CBD is applied to the implementation of various health-related software systems as well.

However, the advantages of using CBD with the current component models used for software system building are not achieved because of the lack of reusability and poor composition mechanisms [6]. To fulfill the ideals of CBD, current research in CBD is focused on new component models and their validation.

Based on the importance of EHRs in NHII and limitations of current software component models, our study will analyze recent problems of implementation for exchanging EHRs with their limitations and give a promising solution for the problems through recently proposed component technologies.
CHAPTER 1: BACKGROUND

This chapter gives a general overview of the main themes in this study: Component-based software development, National health information infrastructure (NHII) and Electronic health records (EHRs). As a research discipline, component-based software engineering is a starting point to understanding this study, thus we consider component-based development in this chapter. NHII and EHRs in NHII are the application areas to apply the software component-based approach. Relevance between the research discipline and the application and motivation of this thesis will be presented as well.

1.1 Component-based development

According to Heineman and Councill’s definition [5], a software component is “a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.” Component-based development (CBD) is the process of building software systems using those components. The elements of construction are summarized as componentization, consume, supply, and manage shown in Figure 1.1 [5]. To build a large system, the system is divided into pieces or subsystems which are implemented as software components (Component Development: componentization), and the components are stored in a component repository. From this repository, the components are supplied, consumed and managed (Component Management). The systems are constructed based on this basis (Solution Development). This development methodology gives advantage of software reuse and reduced time to market, increased reliability of systems, and increased flexibility [5]. To mention today’s component technologies, we should think of component models which define the semantics of components, the syntax of components, and the composition of components. There are many component models: JavaBeans, EJB, CCM, Web services, Koala, KobrA, SOFA, Acme-like ADLs, UML2.0, PECOS, Fractal and etc [6]. These component models are used in current software industry and research.

For a long time, the concept of subroutines was broadly used in programming because using subroutines is good to conserve memory. From this concept, programmers reused the previous subroutines, and useful subroutines are collected in a form of libraries so that a programmer could re-use them easily. This phenomenon caused a paradigm shift in software development and the result of this phenomenon has directly boosted the software engineering discipline [7]. Along the history of programming, researchers introduced programming concepts, methodologies and paradigms based on this phenomenon: pieces of

Figure 1.1: Consume, supply, and manage [5]
programs, information hiding, interfaces, and etc. Object-oriented development was developed from this historical background. Software engineering has accelerated research regarding this paradigm and as a result, important concepts useful in developing software systems such as software architecture and design patterns have been identified and have evolved this paradigm [5]. Finally, CBD has changed the way large software systems are built and realized “Buy, Don’t build” philosophy [5]. Today, the use of commercial off-the-shelf (COTS) components has been achieved to build software.

Current research in this discipline is focused on new component models and evaluation of software components. Lau and Wang surveyed and analyzed current component models, and found that the ideal component models do not still exist according to their taxonomy [6]. It means current software component models should be improved. Several important issues are also mentioned by Lau and Wang: the component design for easy reuse, the design of composition mechanism or theory to enable systematic composition [6]. The issues of component evaluation have come from how appropriate components can be selected [8]. As the software component market grows, the need to evaluate components will also be increased because the overall quality of software systems implemented through CBD is directly impacted by the quality of each software component.

1.2 Electronic healthcare

Information communication technologies (ICTs) are growing, and these technologies have changed health care environments. The outcome of this change is referred as electronic healthcare. Without a National health information infrastructure and its elements, the future of electronic healthcare cannot be certain.

1.2.1 National healthcare infrastructure

The National Health Information Structure (NHII) provides easy access to healthcare data, personal health histories and other clinical information. NHII has made a revolutionary change in eHealth services by providing time saving information to healthcare staff.

The NHII has now grown roots in several EU countries, United States, Canada and Australia. Science researchers are making efforts to improve the efficiency and quality of healthcare services and accelerating health-related research. NHII is a “comprehensive knowledge based network of interoperable systems” which acts like a bridge between public health and medical care [9]. The most important role of NHII is patient’s safety. A complete medical record of a patient is available anytime that helps healthcare individuals to make fast decisions even at remote locations. Patients can also access their medical information.

Information technology is developing rapidly in health care nowadays. Efforts are being made to facilitate people with their medical information and treatment, but at the same time it brings some challenges that need to be resolved in order to maintain the quality of NHII. One of challenges to construct NHII is Electronic Health Record (EHR). The main challenge is the lack of standards and a decentralized system that affects the efficiency and output during interoperability. An improvement in coordination is required between regional and national eHealth networks to maintain the quality of eHealth services [2]. According to Tsiknakis et al., in health information infrastructure, research for software component availability is required for new software development and the integration of existing components to perform new tasks [10]. NHII faces inconsistent quality and security issues. The system needs further development to avoid errors. As the new technologies are expensive nowadays, the cost factor is also a challenge for NHII. It requires more funding in order to meet the stringent standards.
1.2.2  Electronic health records

Electronic health record (EHR) stands for: “a repository of patient data in digital form, stored and exchanged securely, and accessible by multiple authorized users” according to the definition by the International Organization for Standardization (ISO) [4]. EHR has improved the patient safety issues to a great extent. Its implementation has shortened the distance between healthcare professionals. EHR provides accessible and relevant information to the physicians, specialists and also to patients. This information includes patient’s personal information, previous medical history, laboratory tests, medication, treatment and discharge. EHR also provides long term storage capability and can be accessed by large group of people on remote locations at the same time. Healthcare staff can easily update information in a patient’s record that helps the physicians to make reliable and fast decisions.

EHRs play an essential role in clinical research, health services planning and management. Early research for digitized health services planning were focused on patient summary and structured data but nowadays critical research is focused on areas such as standardization, integrated EHRs, architecture of health information systems or networks, and interoperability between existing heterogeneous health information systems have been evolved in the aspects of EHRs [2, 11]. Particularly, exchanging EHRs is a major issue to establish interoperability within NHII and a lot of research has already been conducted and the research studies mentioned about components, their reusability and issues regarding components as future works [10].

During the exchange of electronic health records, there are issues which are a challenge to improve. Physicians are however becoming dependent on such computer systems and this is why more standardized methods are required for the assessment of data quality, time and cost issues. Data “completeness and accuracy” is a critical challenge in exchanging EHR. According to Muller et al. an easy adaptation to new settings that are directly linked by component reusability were mentioned as future work [12]. Based on these scientific evidences, the functional reusability of exchanging EHRs is required to implement NHII due to common industry issues such as development time and cost [2]. In other words, the application area of this study is reuse-based component development to achieve effective and interoperable implementation of exchanging EHRs within NHII.

1.3  Motivation

Most recently developed software systems are constructed by using the component-based approach. This means health-related software systems have also been built through current component models. However, current component models should be improved, and building new component models is a challenging issue as we already discussed in the previous sections. From this fact, although much research about exchanging EHRs have been conducted, we can easily observe that currently proposed outcomes for exchanging EHRs cannot be beyond the limitations such as component reusability of the current component models. In this regard, we can think that component-based development has great relevance to the implementation of exchanging EHRs in NHII because NHII requires the reuse of the function of exchanging EHRs which can solve the recurring problems to exchange EHRs.

Most of the EHR related software and Health Information Systems are implemented by current software component models with lack of reusability. Moreover, current research focusing on practical development is limited to proposing architectures of whole systems or frameworks in a broad view [10, 11, 13, 14]. Considering these facts, we are motivated much to close this gap.
CHAPTER 2: PROBLEM DEFINITION AND GOALS

2.1 Problem Definition

The problem we define in this research is the lack of component reusability in current component models and its blind application to exchanging EHRs. In other words, software components relating with exchanging EHRs or EHR systems in the current research studies are implemented by current component models with the limitation of their reusability. However, the current research applying these models to health informatics area accepts the component reusability without doubt [10, 13].

Electronic Health Records are a challenging area in health informatics, and current research studies related with EHRs show prominent results regarding overall architecture of implemented health information systems or networks, and international standards for EHRs because EHR is a major part of NHII. When NHII based on the robust EHR systems is constructed, it shall impact on improving quality of healthcare services and assisting health-related research studies. In particular, NHII tries to join all health information systems together and requires reuse of the repeating functionality such as exchanging EHRs. In this sense, reusability of software in NHII is very important issue to be resolved.

Research focused on practical development using the specific technologies such as Component-based Development (CBD) are quite few or limited to proposing architectures of whole systems or frameworks in a broad view [10-12]. Component reusability is not adequately achieved in current component models and mostly EHR-related software and Health Information Systems are implemented by these models with lack of reusability. As Component-based Development (CBD) provides increased reuse, less production cost and shortens the time to the market, it is necessary to have components that can be easily reused with "composition mechanisms that can be applied systematically [6]." However, the ideals of software reuse without source code change in current CBD has not been achieved much due to its constraints such as dependency between components [6, 15]. Moreover, the deployment of new clinical information systems and e-health services become easier and faster due to re-usable components, but it still requires more focus on how and when to upgrade the platform itself (considering the cost factor) [11].

In summary, we investigate and validate the recently proposed component models to enhance component reusability and apply the validated component model to exchanging EHRs in order to resolve the issues regarding the component reusability of current component models and its incomplete application.

2.2 Goal and Objectives

The goal of this study is to analyze component reusability of recently proposed component models by applying these models to exchanging EHRs which require component reusability in NHII.

Component reusability is one of the most important benefits of CBD. However, current software component models have not achieved component reusability much [6]. From this fact, we assure that if component reusability of recently proposed component models can be validated by this study, the known benefits of CBD such as reduced time to
market and cost can be readily achieved. Analyzing component reusability will be a basis to select proper software component models to develop various software systems which require software reuse acutely. Particularly, exchanging EHRs is the most important function to construct NHII [2]. When this goal is achieved properly in this study, it will accelerate the challenges in the national scope software development such as NHII.

To fulfill this goal, we define several objectives:

- Identify recently proposed software component models excellent to component reusability.
- Investigate a proper component evaluation method to validate component reusability.
- Evaluate recently proposed software component models by the proper component evaluation method.
- Clarify a performance impact when applying these component models to component development.

When these sub-goals are reached, the goal of this study will be achieved.

### 2.3 Research Questions

To resolve the problem explained in the previous section, we define the following research questions based on the research goal and objectives. Our research focuses on recently proposed component models using exogenous and mediator connectors. Lau et al. proposed exogenous connectors which control all operational calls between components [16, 17]. The mediator connector proposed by Sanatnama et al. controls all operation calls as a compositional operator as well [18]. Using the JavaBeans component model as one of current component models, we study exogenous and mediator connectors in detail in Chapter 4.

1) **What is the better component model to guarantee component reusability for exchanging electronic Health Records between JavaBeans and component models using exogenous connectors or mediator connectors?**

   The first step to the research problem of this study is to identify the appropriate component model or method to enhance component reusability. Because the most common component models still have limitations in realizing component reusability, identifying the appropriate software component model among recently proposed component models is the most important starting point. So, in this part of our research, we shall study recently proposed component models using exogenous and mediator connectors that will be used to analyze which model guarantees component reusability well without code change.

   To answer this question, we focus on prototyping based on selected component models among recently proposed component models and its experimental validation by using component evaluation models. The selected component models are the component model using exogenous and mediator connectors respectively. JavaBeans component model which is one of the most common component models, and the prototypes implemented by the selected component models will be compared with the prototype using JavaBeans component model. The result of this experiment will provide the better component model guaranteeing the component reusability.

2) **How can component reusability be evaluated for systems implemented by using JavaBeans, exogenous connectors, and mediator connectors?**

   The importance of this question is thus; through a proper component evaluation we can evaluate and validate the software component models in terms of reusability. Without answering this question, it is not possible to identify a proper software component model for
exchanging EHRs which requires component reuse. There are several evaluation methods for component reusability and we need to identify the appropriate evaluation method among the recently proposed component evaluation methods. Each method has its own evaluation process. Through literature reviews, we investigate software component evaluation methods and analyze their pros and cons. After that, we can identify the appropriate evaluation method for our study.

The results of component reusability evaluation based on the survey regarding component evaluation methods will be on both theoretical and practical basis to verify component reusability.

3) How big is a performance impact when each component models such as JavaBeans, exogenous connectors, and mediator connectors are adopted for constructing software systems?

During our tests to identify the appropriate component model, there is a possibility for a performance problem to be created even though component reusability is fulfilled. Analyzing this problem is important because component reusability is not the only concern while implementing the function of exchanging EHRs. Other software properties such as performance should be considered as well during the construction of NHII. So, we will observe a performance impact while using these reusable components. This part will be discussed in detail based on the result of analysis for research question 1, 2.

2.4 Expected Outcomes

Meaningful research contributes to knowledge theoretically and practically. New knowledge should change the world and make people happier. How can we contribute to knowledge through this study?

The expected outcomes of this study are as follows:

1) We can confirm which software component model can support component reusability very well.

This expected outcome can make software architects decide which software component model should be used to guarantee component reusability. Depending on the type of development projects, component reusability can be of very high priority. The analysis of recently proposed component models will be useful knowledge to decide the software component model to be used in a project.

2) The result of prototyping for exchanging EHRs will be a critical part of the blueprint to construct NHII and can be used practically in NHII.

During this study, we make several prototypes to which recently proposed software component models are applied. After validating their component reusability, we can clarify which software component model is good for realizing component reusability, and this result will be a robust approach to construct NHII.

3) Scientific validation and analysis results of component models will be an objective, academic basis for software component models in terms of component reusability.

The most important question of this study is how we can scientifically prove if a certain software component model is proper to assure component reusability. For the scientific validation, we use an empirical study and analyze its result. These efforts will establish scientific objectivity of this research.
We demonstrate that these expected outcomes will be enough to contribute to knowledge in software engineering and health informatics area and make this study meaningful one.
CHAPTER 3: METHOD

3.1 Overview

In this chapter, we present how to achieve the goal of our study described in the previous chapter. Both qualitative and quantitative methods are used via following ways: literature reviews, prototyping, and an experiment.

First, we examine current research studies carefully regarding standards for EHRs, software component models, and component evaluation methods through literature reviews. Several standards for EHRs such as HL7 CDA, CEN ENV 13606, openEHR, and DICOM are investigated in this chapter and their implementation technologies are introduced as well. To look at overview of software component models, taxonomy for software component models is discussed before digging deep software component models in detail. Current component models are studied briefly and the comparison of all these models is a basis to select several models to be used in prototyping. Component evaluation methods are very important parts to scientifically validate if the component models to be selected are valuable for implementing highly reusable software components. Secondly, based on the literature reviews, we choose one of standards for EHRs and select several candidate component models for prototyping. Namely, the software component implementation for exchanging EHRs will be progressed. Lastly, the experiment with the results of prototyping will be designed to answer our research questions. For the experiment, we also use one of the proper component evaluation methods and analyze the experiment result through it.

![Overview of research methods](image)

Figure 3.1: Overview of research methods
3.2 Literature review

3.2.1 Standards for EHRs and implementation technologies

Health services have extensive opportunities for improvement through the use of Information Technology applications in different areas, for example in laboratories, patient’s health records and other clinical documentation. In earlier days when Information Technology was growing roots in clinics and laboratories, a patient’s record and the administrative information was kept within the organization. There was no access to the data from outside.

International standards for EHRs such as HL7 Clinical Document Architecture (CDA), International Organization for Standardization Technical Committee 215 (ISO/TC215), and European Committee for Standardization Technical Committee 251 (CEN/TC 251) have a large scope and the current research based on these technologies has brought helpful and positive results regarding overall architecture of implemented health information systems or networks [10].

Some of the common EHR standards and their implementation technologies are shortly discussed in this chapter.

**HL7 Clinical Document Architecture (CDA)**

HL7 CDA is a common XML document markup standard used between independent healthcare-oriented computer systems to exchange all types of clinical information. Based on XML-encoded documents, it is easy to construct formats of standard information and can interchange format and data over the internet. CDA documents are easy to create and reused. It includes almost all sort of multimedia information e.g. sound, text and images.

The HL7 standard was introduced at the start with the 2.x series version. HL7 version 3 is entirely different and is not compatible with version 2. It is challenging to adopt by organizations which are using version 2 because of its complexity. The reason is that the true concepts are limited to only few groups [19].

One of the main purposes of this standard is to enable interoperability between all platforms. It is easily readable at all platforms or devices. The implementation of this technology has reduced the cost factor and has also shortened the distance between healthcare information systems [20].

Some initiatives have taken on the implementation of open source HL7 version 3 to make it more cost effective and easy to apply. Two well known technologies which have been used are: HL7 Java SIG API, developed by a group of Sun Microsystem, Oracle, Kaiser and UNLV, and NCI caAdapter tool, developed by National Cancer Institute, USA [21].

Two basic functionalities of HL7 messaging and parsing are provided by the Java SIG API. A foundation for the future is provided to HL7 version 3 implementations because of its flexibility. Whereas “caAdapter is an open source tool.” Data mapping and transformation is supported by caAdapter from various data sources. Its architecture is based on two components; “the Core Engine and the Mapping Component” where the Core Engine generates and parses messages utilizing HL7 Java SIG API. The Mapping Component gives the mapping capability to HL7 version 2 to HL7 version 3 formats [21].

The standard used for transmitting HL7 messages via TCP/IP is the Lower Layer Protocol (LLP), or known as Minimal Lower Layer Protocol (MLLP) [22]. Based on an
object oriented methodology, HL7 version 3 uses the Reference Information Model (RIM) to create messages, and CDA obtains its contents from this shared RIM and implements in XML. It is a main principle while creating clinical documents that it is compatible with HL7 RIM and XML. Its document specification is transport independent. CDA also includes different sorts of administrative and financial information but its major goal is to give preference to the clinical documents carrying patient care information [20].

**OpenEHR**

OpenEHR provides an open and interoperable platform to health systems for the exchange of EHRs. It is an open standard specification. OpenEHR explains the retrieval, management, storage and the exchange of patient’s health data in EHR. The transmitted information is readable to the computers and these health systems can further process this health information automatically, known as “semantic interoperability”. The system can record and maintain all clinical information including patient history, treatments, test results, imaging and evaluations [23].

The openEHR templates are used by National Health Service (NHS) in the UK as it provides important implementation experience, and it will be used to develop schemas for templates and specifications, also includes addition to “ADL language supporting specialization and templating; Template Definition Model Schema (.xsd of the TDM) and the Template Definition Model (TDM); an object model of template definitions.” [24]. The assessment of a Template Definition generates an Operational Template in openEHR to produce a “single resulting Template” that is corresponding to a single large archetype. Also it is directly used in openEHR systems on runtime. “being the precursor for data capture forms (including using various XML formalisms such as XAML, XForms), and as the input of Template Data Schemas”. This fact facilitates using TCP/IP as “communication protocol.” Template Data Schema (TDS) gives an enhanced capability for integration in openEHR. A TDS is generated for every template with a single transform. The template’s contents are linked as a message to the resulting schema and are appropriate for the “communication and XML data transformation”. The data source generates its contents according to the schemas explaining the result types. The most important fact of TDS approach is that it provides integration and guarantee to any kind of data that “conforms TDS .xsd in the standard XML approach” is converted to openEHR content format [24].

**CEN ENV 13606**

CEN ENV 13606 can be defined as “a subset of the full openEHR specification.” It enables the exchange of records between different health systems [23]. In the beginning there were many challenges in implementation of the first version of 13606 because of the interoperability and compatibility issues between different health systems. In 2002 the pre-standard of 13606 was enhanced to “a full normative European Standard”. This research was based on the concept of “Archetype Methodology” in openEHR “two-level modeling approach”. In an EHR system, CEN 13606 exchanges “EHR Extracts” and further research is being carried out to make these extracts useful for exchanging information between openEHR systems.

ENV 13606 has three types of messages [23]:

- **request ECHR message**: it is sent to retrieve information from receiver.
- **provide ECHR message**: when the sender want to deliver health information to the receiver.
- **ECHR notification message**: It is used to notify about the acceptance or refusal of a request to receiver.
Digital Imaging and Communication in Medicine (DICOM)

DICOM standard was created by the National Electrical Manufacturers Association (NEMA) in 1985 to assist the distribution and viewing of medical images like MRIs, CT scans and ultrasound. This standard is used by hospitals, clinics, imaging centers and specialists. It allows to take faster diagnosis from anywhere in the world. DICOM is common standard for receiving scans from the hospitals. The use of DICOM standard provides a faster and more effective care to the patients. It is used to send their information through the healthcare enterprise. It also covers most image formats for all the medicine.

DICOM is the specification for messaging and communication between imaging machines. It enables users to retrieve images and related information from digital imaging equipment in a standard format using point-to-point connection.

TCP/IP is used as “communication protocol” to communicate between EHR systems that support DICOM file format. “DICOM format” also ensures the image quality while transmission of files as doctors have to make faster and crucial decisions based on these reports. Patients also get better assistance and care from health systems using DICOM [25].

**Composite objects: (Old objects inherited from NEMA)**
- Verification, Storage Query/Retrieve Study Content Notification

**Normalized objects: (New objects defined in DICOM)**
- Patient Management, Study Management, and Results Management.

DICOM standard is used on large scale in following medical specialties:

- Radiology
- Breast imaging
- Cardiology
- Radiotherapy
- Oncology
- Ophthalmology
- Dentistry
- Pathology
- Surgery
- Veterinary
- Neurology
- Pneumology

A single DICOM file contains header and image data. Header stores information about the patient’s name, the type of scan, image dimension, whereas image data contains information in three dimensions. These files can be compressed in different formats e.g. JPEG.

**DICOM Service Elements:**
Using service elements, complex services are built which are called DICOM Message Service Elements or DIMSEs. It can be categorized in Operations (such as “store”) and Notifications (such as “event report”).

**Advantages:**
DICOM provides standard criteria of displaying images, transfer and storage of information. It has improved the cost-effectiveness in health care during interconnectivity between diverse medical systems.
DICOM Standard is used on the following imaging modalities: Computed Radiography, Computed Tomography, Magnetic Resonance, Nuclear Medicine, Ultrasound, and Secondary Capture. It also develops a common framework for all the specific information into a confirmed structure of storing data [26].

3.2.2 Overview of software component models

According to the Heineman and Councill’s definition, a component model plays a role of defining “specific interaction and composition standards [5].” Interaction with other software components is basically accomplished by the provided and required interfaces of components. These interfaces make dependencies between components, and the type of the dependencies is specified by ‘an interaction standard [5].’ Composition standards provide a composition rules which are required when software is constructed by composing components, so the rules decide how components can be assembled. This definition includes interfaces, naming, Meta data, interoperability, customization, composition evolution support, and deployment as important elements of a component model [5]. Namely, a set of standard of these elements constitute a component model.

Lau and Wang extended the concept of a component model defined by Heineman and Councill [5] to the definition of the semantics / syntax / composition of components [6]. The semantics and composition of components are directly related with interaction and composition standards respectively. The syntax of components is represented by component definition language (CDL) which is different from the implementation language or same as it according to each component model [6]. It is a good try for Lau and Wang adding ‘the syntax of components’ as the element of software component model. That is because the syntax of components can impact on not only interoperability between components based on different implementation languages but also component implementation processes. Lau and Wang’s definition gives us the proper basis in understanding the existing software component models in terms of component design, its practical implementation and use. In this section, we look into the taxonomy proposed by Lau and Wang [6], and then investigate major component models which are currently used much through the taxonomy.

**Taxonomy**

Lau and Wang propose taxonomy based on component composition in their paper [6]. This taxonomy categorizes software component models according to composition format in stages of a component life cycle.

![Figure 3.2: An idealized component life cycle][1]

[1]: https://example.com/figure3.2.png
From the fact that system integration in CBD takes place by component composition in ‘several phases of the component-based system life cycle’ [27], Lau and Wang indentified an idealized component life cycle and its 3 phases, the design phase, the deployment phase, and the run-time phase, shown as Figure 3.2 [6, 28]. According to this idealized life cycle, components have the following characteristics for each phase [6]:

- In the design phase, components are produced by builder tools and stored in repository.
- In the design phase, composite components can be formed as subsystems and managed in repository.
- In the deployment phase, components should be used by assembler tools interacting with a repository and make a whole system as a form of binary.
- In the run-time phase, components should be copied and instantiated on run-time environment.

As shown in Figure 3.2, in the design the builder tool generates a composite component through the composition of B and C, and then the composite component ‘BC’ is stored in repository. In the development phase, the composition of A, B, D, and BC for a whole system is conducted by the assembler tool. Finally, the binary components in the deployment phase instantiates as run-time instances. Most of current component models can be categorized by these composition formats in design and deployment phases of the idealized component life cycle.

Lau and Wang categorize 13 major existing component models (JavaBeans, EJB, COM, .NET, CCM, Web Services, Koala, Kobra, SOFA, Acme-like ADLs, UML 2.0, PECOS, and Fractal) based on these composition styles on each phase and define 4 categories for the 13 models [6]:

- Category 1: Design without Repository
- Category 2: Design with Deposit-only Repository
- Category 3: Deployment with Repository
- Category 4: Design with Repository

Based on these categories, we will briefly study these component models with the explanation of each category in the next section.

**Software component models [6]**

Category 1 (Design without Repository) [6] is that components are designed by their scratch and then the composition of component instances is occurred in the run-time phase. There is no flexibility in terms of composition because there is no repository so that the composition of components depends on the design scratches. Actually, services and dependencies of components in this category are represented in the design phase, and they are instantiated in the run-time phase. Acme-like ADL based models such as UML 2.0, PECOS, and Fractal are included in this category.

Category 2 (Design with Deposit-only Repository) [6] is that the components the design phase are deposited in a repository and then instantiated in the runtime. But those components in the repository cannot be retrieved and as a result, the composition is occurred in the run-time phase and same as the composition in the design phase. The difference with Category 1 is whether the repository exists or not. The component models used in COM, EJB, .NET, CCM, Web Services belong to this category.

Category 3 (Deployment with Repository) [6] is that the components in the design phased are stored in a repository but the retrieval of components in the repository is occurred in the deployment phase. It means there is no composition in the design phase but in the deployment phase. After composition in the deployment phase, the component instances are executed in the run-time phase. JavaBeans is the only component model between 13 component models. In Java, components stands for beans, which are deposited in the
ToolBox (a repository) of BDK, deployed to BeanBox (an assembler) by dragging Beans from the ToolBox, and executed in the run-time environment.

Category 4 (Design with Repository) [6] has a distinctive property comparing with Category 2. That is the components can be retrieved in the repository. For example, in Koala, three types of connectors for components (that is binding, glue code, and switch) are defined as well as components and composite with the components from the repository in the design phase. Namely, by using these connectors, composite components can be generated in the design phase and deposited in the repository. There is no composition in the deployment phase and component compositions are same as the compositions in the design phase. This category includes Koala, SOFA, and KobrA.

From this taxonomy, Lau and Wang summarize 13 component models shown as Table 3.1 based on the idealized component life cycle. We can notify Category 4 mostly fulfills the idealized component life cycle and any category except for Category 3 doesn’t fulfill composition in the deployment phase. So, Lau and Wang insist the current component models should be improved for achieve the idealized component life cycle and mentioned the ideal category ‘Design and Deployment with Repository’. In the next chapter, we will investigate this gab via recently proposed component models.

### Table 3.1: A taxonomy based on composition [6]

<table>
<thead>
<tr>
<th>Category</th>
<th>Models</th>
<th>Design</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design without Repository</td>
<td>Acme-like ADLs, UML 2.0, PECOS, Fractal</td>
<td>Deposit-N: x, Retrieve: x, Compose: ✓</td>
<td>Deposit-C: x, Compose: ✓</td>
</tr>
<tr>
<td>Design with Deposit-only Repository</td>
<td>EJB, COM, .NET, CCM, Web Services</td>
<td>✓, x, ✓</td>
<td>✓, x, ✓</td>
</tr>
<tr>
<td>Deployment with Repository</td>
<td>JavaBeans</td>
<td>✓, x, x, x</td>
<td>✓</td>
</tr>
<tr>
<td>Design with Repository</td>
<td>Koala, SOFA, KobrA</td>
<td>✓, ✓, ✓, ✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

* Deposit-N: new components can be deposited in a repository.
* Retrieve: components can be retrieved from the repository.
* Compose: composition is possible.
* Deposit-C: composite components can be deposited in the repository.

#### 3.2.3 Overview of software component evaluation methods

Component-based development (CBD) and Commercial off-the-shelf (COTS) components are general in current software system industries due to their benefits such as component reuse. However, one of the challenging issues has been how to evaluate and select robust components for their reuse and many research studies regarding component evaluation have conducted [8, 29-32].

To resolve this issue, Ismail et al. reviewed current component evaluation approaches and categorized them into 4 groups in terms of software reuse: Product Line Engineering Components (PLC), Originality Components (OC), Quality Components (QC), and Reusable Components (RC) [29]. OC and QC approaches mainly focus on components’ quality rather than reusability. PLC and RC approaches, however, aim at evaluation of core assets in software product lines and component itself respectively. Although to define these groups
Ismail et al. use three criteria (scope, technique, and level of validation) which are partial members of a evaluation framework proposed by Goulão and Abreu [29, 33], comparison result between the 4 groups based on three criteria shows PLC approach fulfills three criteria as higher weight than other approaches [29], followed by RC approach. Of course, this review paper mentions all approaches have a room for improvement in terms of ‘independently validated’ which is a subcategory of level of validation [29]. However, the result of comparison between all approaches shows comparative maturity of component evaluation approaches [29]. In particular, for our clarification regarding implementation of exchanging EHRs as a core asset in NHII based on the view of software product lines, the review of Ismail et al. [29] is useful to this study in terms of deciding the proper component evaluation method even though current evaluation approaches are required to be improved. Namely, PLC and RC approaches will be used in this study in terms of reusability of the core asset and its software components. Detail evaluation framework setting of these approaches for the experiment will be discussed in the chapter 5.

3.3 Prototyping

To realize high component reusability for exchanging EHRs, we will investigate certain EHR standards, related implementation technologies, and recently proposed component models deeply in theoretical work. After this investigation, we can identify pros and cons for each investigation result and then decide how to conduct prototyping. Basically, we implement 3 prototypes for exchanging EHRs. These prototypes are the combinations between EHR standards, related implementation technologies, and three candidates of component models. Namely, prototyping steps are as follows:

- Choose and investigate the appropriate standard for EHRs and related implementation technologies.
- Choose and inquire into 3 candidate component models in detail.
- Prototyping
  - Design components for exchanging EHRs by using the selected EHR standard as core asset in NHII
  - Implement components by using the selected software component models and implementation technologies.

As we clarify, the functionality for exchanging EHRs will be prototyped as core asset in NHII. Based on software product line, this core asset will be constructed according to core asset development of software product line activities [34].

3.4 Experiment

Experiment is a core part of this study. That is because the goal of this study, analyzing component reusability, can be directly achieved by this experiment with the result of prototyping.

For the experiment, we define the selected 3 components models as independent variables. Namely, each prototype based on one of the selected component models is an independent variable. According to each prototype, the components’ characteristics such as functional commonality, non-functional commonality, variability richness, applicability, and tailorability will be affected as dependent variables. We will use the evaluation framework and its metrics for reusability of core assets proposed by Her et al. [32] as PLC approach. The attributes of this metrics are same as the characteristics used as dependent variables. For RC approach, the metrics suite for measuring reusability of software components proposed by Washizak et al. [35] will be applied. After evaluating components by this experiment, we
analyze the experiment result and can draw the conclusion regarding the highly reusable component model and its implemented components for exchanging EHRs.

In the next chapter, the theoretical work, we will investigate the EHR standard and implementation technologies we will use in the experiment. Underlying theories of recently proposed component models chosen for the prototyping will be presented as well.
CHAPTER 4: THEORETICAL WORK

This chapter explains underlying theories of recently proposed software component models as well as the chosen EHR standard and implementation technologies for the experiment in detail. Through this chapter, we can understand how these concepts can be related with component reusability for exchanging EHRs. Theoretical bases of them yield robust results for this study.

4.1 The chosen EHR standard and its implementation technologies

Among EHR standards introduced in Section 3.2.1, we chose HL7 CDA for the model of our research. CDA (Clinical Document Architecture) has been developed by HL7, which is an international standardization organization approved by American National Standards Institutes (ANSI), to share and exchange clinical data. CDA not only covers vocabularies and relationships of clinical data comprehensively but also is very flexible in adding new variables. Thus, it is widely accepted that all the contents in any clinical document can be expressed with CDA [36].

In this section, we first explain the general framework of HL7 CDA and describe the procedure in constructing HL7 CDA documents in detail.

4.1.1 General framework of HL7 CDA

HL7 suggests Reference Information Model (RIM) to define basic classes of medical data and the hierarchy of them. RIM creates medical documents and messages composing those classes and upper level domains [37]. Six main classes in RIM are as follows:

*Act class*: This class defines past, current, and future acts that are required or intended for patients. For example, medical examination, operation, medical treatment, and patient education are included in this class.

*Entity class*: This class contains information about persons or institutes that conduct acts.

*Participation class*: This class connects entities and acts by indicating the role of an entity in an act.

*Role class*: This class defines a type of role that is verified and approved for an entity in an act. For example, the role of an anesthetist among entities who are participated in an operation is putting a patient under anesthesia. This information is written in the Role class of the entity.

*ActRelationship class*: This class represents the relationship between two consequent acts. If a patient has an operation for cholecys-tectomy as the patient’s case has been diagnosed as cholelithiasis, an act ‘diagnosing the case as cholelithiasis’ and an act ‘operating on the patient for cholecys-tectomy’ are linked each other as a cause and a result with the ActRelationship class.

*RoleLink class*: This class links two related roles. If an entity takes roles of ‘hiring’ and ‘allocating jobs’, and the role of ‘allocating jobs’ is only valid for this entity when the entity takes a role of ‘hiring’, this dependency between two roles are explained in the RoleLink class.
RIM defines almost 70 basic classes for HL7 messages. Domain Message Information Model (D-MIM) is a subset of RIM. D-MIN consists of classes, attributes and relationships which express those messages of a specific field. Furthermore, Refined Message Information Model (R-MIM), which is instantiated from D-MIN, is used for representing particular information of a particular message in more confined groups.

CDA R-MIM, as a subset of RIM, is developed for describing clinical information [38]. CDA R-MIN inherits all the properties of RIM. In addition, CDA R-MIN contains the class cardinality to define necessary classes and the number of appearance of them in a CDA document.

CDA uses 22 data types with 4 classifications, which are Abstract type, Basic type, Generic collections, and Timing Specification. For Abstract type, CDA uses ANY type, which is the highest level of data types and defines basic attributes that are shared for all the data types. For Basic types, Boolean (BL), Encapsulated Data (ED), Character String (ST), Character String with Code (SC), Postal Address (AD), Entity Name (EN), Person Name (PN), Organization Name (ON), Concept Descriptor (CD), Coded with Equivalents (CE), Coded Simple Value (CS), Instance Identifier (II), Telecommunication Address (TEL), Integer Number (INT), Ratio (RTO), Physical Quantity (PQ), and Point in Time (TS) are used. For Generic Collections, Set (SET), Interval (IVL), and Sequence (LIST) are used. For Time Specification, General Timing Specification (GTS) is used. The overall hierarchy of these data types is depicted in Figure 4.1. BN, BIN, ADXP, ENXP, CV, URL and QTY in Figure 4.1 are not those data types used in CDA but intermediate concepts in the hierarchy.

Medical vocabularies used in CDA include Logical Observation Identifiers Names and Codes (LOINC) and SNOMED Clinical Terms (SNOMED CT) as well as those are defined by HL7.

LOINC is the set of vocabularies that cover all sorts of medical examinations, treatments, experiments and diagnosis conducted in hospitals. Using LOINC, hospitals can exchange and utilize medical data conveniently. LOINC 2.16 released in December 2005 contains 42,499 clinical terms. Regenstrief Institute manages these data [40]. When a HL7 CDA document is created, an appropriate LOINC code, which indicates the type of the document, is searched considering seven LOINC fields: Component, Property, System (Sample) Type, Type of Scale, Type of Method, Class, and Status. First, we search a text
value of the required document type in the Component field. If the Status field is DEL, exclude the record from candidates. Then, we check if the System (Sample) Type and Type of Method are right for the searching document type. The Type of Scale is referred to check if the document classification is appropriate. After checking the Property and Class fields, the LOINC code fit for the purpose can be finally confirmed [39].

SNOMED CT is a standard medical terminology structure developed by College of American Pathologists and National Health Service of UK in 1999. This is known as a comprehensive set of clinical reference terminologies. SNOMED CT released in January 2006 covers 366,170 concepts, 993,420 descriptions and 1,460,000 relationships. A special feature of SNOMED CT is that it allows users post-coordination, by which users can construct various new concepts by composing more than two existing concepts. As illustrated in Figure 4.2, the relationship type links domain and value and the combination of them makes a new SNOMED CT code. However, the post-coordination has drawbacks that the same concept may be expressed differently by users using different compositions and it may confuse other users [41].

![Figure 4.2: The concept of post-coordination in SNOMED CT schema [39]](image)

Clinical documents written with LOINC and SNOMED CT help doctors and other clinicians in sharing clinical information and in accessing to clinical knowledge database easily.

4.1.2 Creation of a HL7 CDA document

CDA consists of two parts: header and body[39]. CDA header contains critical information that enables sharing and managing patients’ clinical documents consistently among hospitals and doctors. Contents in CDA header include the clinical document class, the participants class group, and the act-relationship class group. The clinical document class describes the information about this clinical document. Mandatory attributes of this class are a unique document identifier (id), type of the document (code), creation time of the document (effectiveTime), and confidentiality (confidentialityCode). The participants class group contains information about patients (service targets) and medical personnel (service actors). This class group includes the author class, legal authenticator class, authenticator
class, custodian class, and record target class. The act-relationship class group includes the parent document class, which indicates the previous documents, the related document class, which shows relationships among documents, the service event class, which describes acts recorded in the document, the documentation of class, which links the service event classes, the order class, which contains orders, and the in fulfillment of class, which connects the order classes.

CDA body is composed of either the non-XML body class or the structured body class. The non-XML body class is used for referring to outside data encoded with other data format than XML. The structured body class is used for a XML format document. This class includes several section classes. A section class contains a unique identifier of this section (id), type of the section (code), title, text, confidentiality, and language code. Texts of the structured body class require the entry structure, which enables computers to interpret the text. Contents of the text are written in codes of a standard terminology such as SNOMED CT in the entry part. The entry structure is based on HL7 Clinical Statement model.

Kim [39] explains the detailed procedure creating a HL7 CDA document by using a sample clinical document, which is a discharge summary note. Records of a discharge summary note include the patient’s registration number, name, age, sex, department, hospitalization date, discharge date, discharge disposition location, diagnosed disease, general history and plan of treatment.

At the first step, those recorded elements are classified into header and body. The patient’s registration number, name, age, sex, hospitalization date, discharge date, discharge disposition location and department are categorized into header, while the diagnosed disease, general history and plan of treatment belong to body. Besides these elements, the confidentiality, author, and administrative institute of this document are necessary in a CDA document. Sample mapping results of the header part of discharge summary note on the CDA structure are displayed in Table 4.1.

<table>
<thead>
<tr>
<th>Table 4.1: Sample CDA header for discharge summary note elements [39]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDA Element Name</td>
</tr>
<tr>
<td>ClinicalDocument</td>
</tr>
<tr>
<td>code</td>
</tr>
<tr>
<td>title</td>
</tr>
<tr>
<td>effectiveTime</td>
</tr>
<tr>
<td>recordTarget</td>
</tr>
<tr>
<td>patientRole</td>
</tr>
<tr>
<td>id</td>
</tr>
<tr>
<td>Patient</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>administrativeGenderCode</td>
</tr>
<tr>
<td>birthTime</td>
</tr>
<tr>
<td>encompassingEncounter</td>
</tr>
<tr>
<td>dischargeDispositionCode</td>
</tr>
</tbody>
</table>

Second, appropriate LOINC codes for each element are matched. Sample LOINC codes for discharge summary note elements are listed in Table 4.2.
Table 4.2: Sample LOINC codes for discharge summary note elements [39]

<table>
<thead>
<tr>
<th>LOINC_NUM</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>34105-7</td>
<td>Discharge summarization note</td>
</tr>
<tr>
<td>11535-2</td>
<td>Hospital discharge DX</td>
</tr>
<tr>
<td>11329-0</td>
<td>History general</td>
</tr>
<tr>
<td>18776-5</td>
<td>Plan of treatment</td>
</tr>
<tr>
<td>42345-9</td>
<td>Discharge functional status</td>
</tr>
</tbody>
</table>

Third, the elements of discharge summary note are converted to a CDA document with the XML file format. Figure 4.3 shows the body of the completed HL7 CDA document for the sample discharge summary note. As this CDA document is written in XML language, the body part shows the structured form. The structured body is composed of two levels. The first level, which is enclosed with `<text>` and `</text>`, is for human to read and understand the contents. The next level, which is placed between `<entry>` and `</entry>`, is expressed as SNOMED CT codes that machines interpret to take an appropriate action.

Figure 4.3: Sample CDA body for discharge summary note [39]

### 4.2 Underlying theories of recently proposed component models

As Lau and Wang drew in their study, current software component models have a room for improvement in terms of composition in both design and deployment phase, and have critical issues as well: how to design readily reusable components and how to design component mechanism for systematic composition [6]. Without proper composition between components, their reusability can not be achieved so that it is required these issues should be definitely resolved. Most of current component models use indirect or direct message passing for composition mechanism which make it difficult to reuse components because of tight coupling between components and composite components, and do not have composition theory [6]. Although some models such as ACME-like ADLs which have
architectural units as components have a simple composition theory using ports in software architectures for composition, they still have problems for systematic composition because this simple theory just defines the level of ports but not the level of a component itself [6]. In the past before Lau and Wang clarify these problems, studies such as component adaptation and dependency injection had conducted for components’ reuse, but there were limitations in terms of feasibility and reusability and it was focused on other factors rather than component reusability [15].

In this section, we investigate three component models recently proposed by Lau and Wang [42], Sanatnama et al. [18], and Lim et al. [15]. Particularly, we explain the core concepts of their models which stand for ‘Exogenous connectors [42]’, ‘Mediator connector [18]’, and ‘Active binding technology [15]’ respectively.

4.2.1 Exogenous connectors

As composition operators, exogenous connectors in component model proposed by Lau and Wang [42] are a kind of coordinators to control interactions between components. In this model, components do not call methods of other components but the methods are only invocated by responsible exogenous connectors.

Actually, the concept of connectors is used in traditional ADLs to represent interaction among components, and a component passes their controls to other components via connectors. This case where connectors play the role of just a path for interacting between components make components be tightly coupled while compositing and as a result, components can not avoid depending on other components. On the contrary to the connectors in traditional ADLs, exogenous connectors play the role of a controller to actively manage the interaction between components rather than just an interaction path.

To specifically compare the connection via exogenous connectors with the connection among components in current component models such as EJB and ADLs, we can think of direct and indirect message passing. Figure 4.4 clearly shows two types of message passing in order to invoke methods of components. Most of component models meet one of two types [43]. Direct message passing shown in Figure 4.4 (a) is that components directly call methods of other components, and component models such as EJB, COM, CCM, UML2.0 and KobrA follow this mechanism [43]. Without connectors, these models directly interact with other components. The second type, indirect message passing, uses connectors for interaction between components. For example, shown in Figure 4.4 (b), the component ‘A’ calls the method ‘a()’ of the component ‘B’ via the connector ‘K1’ indirectly. JavaBeans and several models using ADLs or ADL-like languages follow this indirect message passing to invoke other components’ methods. Regardless of the difference between two types, both direct and indirect message passing coordinate the interaction controls to other components through a component itself and as a result, the composition between components is adhered tightly. This tight coupling of components hampers the easy reuse and systematic composition of components.

![Figure 4.4: Connection by message passing [43]](image)
To decrease the degree of coupling when components are composed, Lau and Wang proposed exogenous connectors [42]. As shown in Figure 4.5 (a), the direction of method calls is from connectors to components on the contrary to messaging passing mechanism shown in Figure 4.4. This mechanism using exogenous connectors makes possible for components to encapsulate their control flow. Namely, control flows in connectors are hided shown in Figure 4.5 (b). This means computation logics of components are clearly separated with controls so that components are decoupled with other components.

Additionally, for using exogenous connectors in component models, a type hierarchy of exogenous connectors should be considered. In this model, the whole system must be implemented by connections of exogenous connectors themselves for complete control structure for the system as well as connections between components and exogenous connectors [43]. Before explain the type hierarchy, we need to glance definitions of components in this model.

**Components [43]**

Lau and Wang listed two definitions of a component based on exogenous connectors [43]:

- “Definition 1. A *software component* is a software unit with the following defining characteristics: (i) encapsulation and (ii) compositionality.”
- “Definition 2. An *atomic component* $C$ is a pair $<i,u>$ where $u$ is a computation unit, and $i$ is an invocation connector that invokes $u$’s methods. $i$ provides an interface to the component $C$.”

As we can see Figure 4.6, an atomic component consists of an invocation connector and a computation unit. The invocation connector plays the role of calling methods in the computation unit and getting its result. In particular, a composite component is completed by several atomic components and its composition connector shown as Figure 4.6 (b).
Figure 4.7 clearly shows two characteristics of ‘Definition 1’: encapsulation and compositionality. In Figure 4.7 (a), we can see the invocation connector (IU) can call only methods in the computation unit (U) because of encapsulation. Likewise, the composite component shown in Figure 4.7 (b) is encapsulated with the composition connector (K) and several atomic components (C1, C2,…, CJ). This means both an atomic component and a composite component are independent and decoupled so that they can be readily reused and composed.

Types of exogenous connectors [42]

Types of exogenous connectors can be distinguished by their levels where components and connectors are connected. At the first level, an invocation connector takes a computation unit shown as Figure 4.6 (a). As a unary operator, the invocation connector invokes methods of the computation unit. Figure 4.6 (b) shows the composition connector (K) at the second level which is a n-ary connector where n is J, for connecting invocation connectors. Likewise, a whole system is composed by components and various connectors with hierarchy and Figure 4.8 shows the hierarchy of three levels, and this hierarchy can be extended further.

![Figure 4.8: Hierarchy of exogenous connectors [42]](image)

Lau et al. defined connectors by this hierarchy in terms of the number of levels [42]:

<table>
<thead>
<tr>
<th>Basic types</th>
<th>Component, Result;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connector types</td>
<td>L1 ≡ Invocation ≡ Component → Result;</td>
</tr>
<tr>
<td></td>
<td>L2 ≡ L1 × ... × L1 → Result;</td>
</tr>
<tr>
<td></td>
<td>L3 ≡ L × ... × L → Result</td>
</tr>
</tbody>
</table>

Where L is either L1 or L2;

... According to this definition, we can be notified that from the second level (L2), the connectors have variable arities. Moreover, from the third level, connectors are polymorphic so that connector types can be shown in various formats on the contrary to that in the first and second level connectors are connected with components and the connector L1s respectively. To generalize this definition with an arbitrary number n of levels, Lau et al. also defined the type hierarchy [42]:

\[
\begin{align*}
L_1 & \equiv \text{Component } \rightarrow \text{Result} \\
L_2 & \equiv L_1 \times \ldots \times L_1 \rightarrow \text{Result} \\
For \ 2 < i \leq n, \quad L_i & \equiv L(j_1) \times \ldots \times L(j_m) \rightarrow \text{Result}, \text{ for some } m \\
where \ j_k & \in \{1, \ldots, (i - 1)\} \text{ for } 1 \leq k \leq m, \\
\text{and } L(i) & = \begin{cases} \\
L_1, & i = 1 \\
\vdots \\
L_n, & i = n.
\end{cases}
\end{align*}
\]

4.2.2 Mediator connector

Sanatnama et al. proposed a component model using mediator connector as a composition operator [18]. The idea of the mediator connector is derived from the concept of mediator pattern for objects’ decoupling which is one of design patterns proposed by Gamma et al.
[44], and the mediator connector is used for defining the relation between components [18]. However, the main difference is that the mediator pattern still causes tight coupling between objects because the origin of an object’s control is the object itself but not a mediator object [18]. On the contrary to the mediator object, the mediator connector generates and manages components’ controls by itself and as a result, loose coupling can be achieved.

In this model using mediator connector, the relation between components can be expressed by interactions. Sanatnama et al. defines an interaction as a set of activities in components such as a sequence of method calls [18]. Figure 4.9 shows an interaction diagram describing a bank system using the mediator connector. As we can see Figure 4.9, there is no interaction between components. Instead, components interact with each other through the mediator connector. Figure 4.10 shows an overall process to build a system using the mediator connector. The first step in this process is designing an interaction diagram, and then, based on this diagram, a kind of description document which is a so-called attachment is made and used for descriptively defining component interactions between moderator connector and components. We can be aware of potential usage scenarios of the system from the description of interactions in the attachment, and this description makes possible to probe the logic of an operation, function and method as well [18]. For the next step, mediator connector interprets the attachment, and the whole or sub system is constructed by composing all components and connections according to this attachment. In the run-time phase, the mediator connector initiates method calls and manage their results from components.

Figure 4.9: Interaction diagram using mediator connector [18]

Figure 4.10: The process of building up a system using attachment and mediator connector [18]
Implementation of mediator connector

Figure 4.11 shows the mediator connector diagram designed by Sanatnama et al. [18]. To implement the mediator connector as a framework, Sanatnama et al. declare typical objects for interaction descriptions which are interaction, components, messages (method calls), and parameters (in/out) [18]. The mediator connector design of Sanatnama et al. [18] is based on these typical objects for interaction descriptions. In the paper of Sanatnama et al., the mediator connector is implemented in Java programming language and components are semantically defined Java class. As we could see the interaction diagram as shown as Figure 4.9, components do not invoke other components’ method but their methods are invoked by the only mediator connector. For the multiple objects such as interactions, components, method calls, and parameters, each object is implemented as a factory class as shown in Figure 4.12. For example, if methods calls repeat many times and a related component is instantiates repeatedly as well, the performance of the system using mediator connector will be decreased. To avoid this problem, Sanatnama et al. used a flyweight component which shares data with other similar components. If the mediator connector requires to instantiate a certain component, the mediator connector requests the component to the factory. If the factory has the flyweight component which fit to the required component, it returns the reference of the component and the mediator connector can use the required component. Moreover, since this model uses Java reflection, the methods of a component can be invoked dynamically [18]. The main advantage of this model using the mediator connector is an attachment which is used as the composition operator. This mechanism gives the easy composition of independent components and the flexibility to compose components. This flexibility positive effects on the component reusability as well.

Figure 4.11: Mediator connector design [18]
Components Interaction Markup Language for mediator connector [45]

In the further work of Sanatnama et al., Component Interaction Markup Language (CIML) is proposed for an attachment which is the description document for components’ interactions [45]. CIML make it possible for the formal representation of interaction diagrams to map to existing programming languages [45]. In other words, CIML should be language-independent to define component interactions [45]. As shown in Figure 4.10, an attachment for describing components’ interactions is written in CIML. CIML follows the XML 1.0 specification so that CIML scripts can be available for software tools as well as software programmer [45].

Table 4.1 shows CIML elements to define components’ interactions. Based on these elements, an example CIML instance document is described in Figure 4.13.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciml</td>
<td>Indicates the start of a CIML document</td>
</tr>
<tr>
<td>Contains</td>
<td>Declares a list of existing components</td>
</tr>
<tr>
<td>Initialize</td>
<td>Declares a component</td>
</tr>
<tr>
<td>InitCall</td>
<td>Declares a list of initiating method calls</td>
</tr>
<tr>
<td>Interaction</td>
<td>Declares an interaction in the system</td>
</tr>
<tr>
<td>OperationCall</td>
<td>Declares a method call</td>
</tr>
<tr>
<td>Inparameter</td>
<td>In-parameters to the method</td>
</tr>
<tr>
<td>Outparameter</td>
<td>Return value from a method call</td>
</tr>
</tbody>
</table>

As we can see the document for the small bank system in Figure 4.13, CIML represents component declaration, initialize declaration and interaction declaration. Although CIML has a room for improvement in order to compose components in the design phase, it improved an attachment concept proposed by Sanatnama et al. [18]. The main improvement for the attachment is that CIML gives a generic framework defining components and their interactions. This merit allows mediator connectors to be applied in various development environments as a generic way defining components’ interactions when composed.
4.2.3 Active Binding Technology

Active Binding Technology (ABT) is a new component model proposed by Lim et al. [15]. This model tries to solve the component assembly issue, the independent component assembly from component development [15]. If the component assembly can be possible regardless of the component development, easy reuse of binary components will be supported by independent component assembly. In this regard, Active Binding Technology wraps up a component assembly method and a runtime environment as well as a new component model.

The main motivation of this proposed component model is that the current component models we already have gone through in the former chapter do not support the reuse of binary components [15]. It means changing components’ code is needed for reuse. Namely, the main concern of this model is focusing on how to match different interfaces between existing business components developed by third parties [15]. The following list represents main characteristics of ABT [15]:

- Software development process is divided into two phases: component development and component assembly.
- ABT allows writing independent business components regardless of component dependency between other components.
- The assembler composes business components by using glue components which adapt mismatched component interface. (Interface Mediation) Through the assembler, component’s interfaces can be bound actively even the interfaces are mismatched each other. (Active Binding)
- The method and tool of adapting interface mismatches is supported by ABT

Lim et al. also defined three tenets supporting component reuse summarized as follows [15]:

- Tenet1: each software component should be developed as a complete independent
part which can be reused in the binary form.

- Tenet2: components should include important information such as the definition of provided and self-defined required interfaces, and their metadata. This information is a basis for binding each component in the form of black boxes.
- Tenet3: New component models should be supported in the later commercial component technologies.

Based in these tenets, a new component model, a component assembly method, a runtime framework, and an automation tool were proposed as a name of Active Binding Technology.

**Active Binding Component Structure and implementation**

In the current component technologies based on JAVA or .NET, components’ interactions are implemented with tight coupling. As shown in Figure 4.14, the client component dependents on the provided interface of server component. In this case, it is not easy to independently reuse the client component without the server component because of its dependency on the server component. Moreover, the client component contains the code invoking the provided interfaces in the server component so that if the client component is composed with the other server component which has the different name of the provided services, the code change could not be avoidable in order to reuse the client component with the other server component.

![Figure 4.14: Physical model of component dependency](image)

To overcome this issue, Lim et al. proposed the Active Binding component structure shown as Figure 4.15 [15]. The unique difference comparing with Figure 4.14 is that a component has required interfaces as well as provided interfaces within the component. These ‘self-defined’ required interfaces within a component are used for active invocation to the provided interfaces of other components in contrast to passively calling the provided interfaces of the server component shown as Figure 4.14. This component structure makes a component as a ‘complete part’ [15] with no dependency on other components. In this regard, we can say this structure fulfills the tenet 1 mentioned above.

![Figure 4.15: Implementation Structure of Active Binding Component](image)

Figure 4.16 shows a component file structure using ABT in .NET. This structure also includes a glue component identifier and required interface metadata as extended parts of .NET component parts. This additional information of ABT is used for generating a glue component which supports assembling active binding components with no code change. The detail process of active binding component assembly will be described in the next part.
Active Binding Component Assembly

The key factor of active binding component assembly is a glue component which is a mediator coordinating interactions between components. As we already shown in Figure 4.16, the glue component is generated by manifest and type metadata of an active binding component in the component file without any code change of each component [15]. Figure 4.17 shows how active binding components are assembled by a glue component. If the required interfaces of an existing client component are not matched with the provided interfaces of an existing server component, the developer can match the required interfaces to the corresponding provided interfaces of the server component through adding adjusting code (glue code) to the glue component. By this way, both syntactic and semantic discrepancy between components can be overcome. Namely, the glue code resolves not only the difference between the defined name of the required interface of the client component and the provided interface of the server component but also the parameter of return values of the components.

Until now, we investigated the underlying theory basis of the recently proposed new component model: exogenous connectors, mediator connectors, and active binding technology. The component models proposed by Lau et al. [43] and Sanatnama et al. [18] are that the control flow between components is originated from exogenous connectors or mediator connectors respectively. This separated control flows of components make possible
for the components to be loosely coupled each other. However, although the control flow between components in ABT proposed by Lim et al. [15] is still originated from the components, it defines required interfaces so that the component can be an independent part from the target component. Based on this each characteristic of these 3 new component models increases increasing the component reusability, we need to verify which model highly realizes the component reusability.

In the next chapter, we will make 3 prototypes based on these new component models and conduct an empirical study. However, we do not implement the prototype using ABT because ABT supports the component reusability with the different way, matching components’ interfaces, when comparing with exogenous and mediator connectors. For the accuracy of the experiment, we will implement the prototypes which can be explained in the view of the easy composition way. Thus, we will use one of common component models (JavaBeans), the exogenous connectors, and the mediator connector for the prototyping.
CHAPTER 5: EMPIRICAL STUDY

The goal of our study is focused on the component reusability analysis of recently proposed component models. To achieve this goal, it is important to validate whether the component models which we look into the previous chapter are proper to developing the highly reusable components or not. Thus, this chapter describes the overall experiment process to validate the component models: prototyping, and the reusability evaluation of the prototypes.

5.1 Prototyping

In this section, we design the prototypes which are called ‘EHR exchanger’ used in the experiment. The reusable components in the prototypes are implemented for exchanging EHRs as the consisting parts of the EHR exchanger.

5.1.1 Basic framework of exchanging EHRs

To clarify the scope of the prototype implementation, we define a basic framework of exchanging EHRs using the Clinical Document Architecture (CDA) shown as Figure 5.1.

![Figure 5.1: Basic Framework of Exchanging EHRs using CDA](image)

To exchange EHRs as the form of a clinical document, minimal basic operations are described as the following:

- Generating a clinical document by clinical data from EHR system.
- Sending the clinical document to the third-party systems such as Health Information System (HIS) and EHR systems of other hospital or institutes.
- Receiving the clinical document generated by third-party systems.
- Parsing the clinical document into EHR data and Storing the data in the EHR system.

These basic operations are mapped to Clinical Document (CD) generator, Sender, Receiver, and Parser respectively shown in Figure 5.1.

5.1.2 Scenario for the prototype implementation

For the prototype implementation, we assume these situations:

- Hospital A and B have their own EHR systems, and all medical or health records of patients in the hospitals are stored in the corresponding EHR Systems respectively.
• The hospital A makes the patient A move to the hospital B because the patient A should be operated by health care providers of the hospital B.
• Health care providers in the hospital B request the discharge summary of the patient A to the hospital A.
• The discharge summary of the patient A generated from the EHR system of the hospital A is transferred to the EHR system of the hospital B.
• The transferred information of the discharge summary is used for the patient A’s continuous care by health care providers of the hospital B.
• After the large operation of the patient A, the patient A moves from the hospital B to a recovery room in the hospital A.
• Health care providers in the hospital A also requests the discharge summary of the patient A to the hospital B in order to continue to care for the patient A.
• The discharge summary of the patient A generated from the EHR system of the hospital B is transferred to the EHR system of the hospital A.
• The transferred information of the discharge summary issued by the hospital B is used for the patient A’s continuous care by health care providers of the hospital A.

The discharge summary including summarized records of individual inpatient care is one of important clinical documents and the most commonly shared information between health care providers in hospitals [12, 46]. Figure 5.2 depicts the scenario transferring the discharge summaries between the hospital A and B to share the important patient’s information.

![Figure 5.2: Transferring The Discharge Summaries between Two Hospitals.](image)

5.1.3 Prototype 1: Basic implementation using JavaBeans

To compare the prototype implementation using the current component model with the implementation using recently proposed component models, we developed Prototype 1 using JavaBeans model which is the component model for Java application. Considering WebService is mostly used technology for interoperability between heterogeneous systems, we implement the part of CD receiver as deployable Java Web application with WebSevice. Figure 5.3 shows practical implementation details of Prototype 1 developed on Java EE platform. The CD Generator consists of Generator and Mapper JavaBeans. Generator bean plays the role of generating CDA documents and Mapper bean supports Generator bean by providing the mapping information between data from EHR system and attributes of specific clinical documents. Sender bean transmits the generated CDA documents, and Connection Adapter gives the way to communicate with other EHR exchangers. WebService is used as communication technology in this study. For the CD receiver, Receiver component is implemented by a WebService module ‘Receiver’ in Java web application. The CD parser
interprets the received CDA documents and split the interpreted data into meaningful information to be able to map with attributes of clinical documents.

5.1.4 Prototype 2: Implementation using exogenous connectors.

As exogenous connectors encapsulate control and data flow between components, they are used as composition operators between components [42]. Lau et al. are developing this component model to the mature stage. Thus, in this study we just use preliminary implementation using Java reflection for prototyping [6, 42, 43, 47].

Figure 5.4: Architecture of EHR Exchanger using exogenous connectors

Figure 5.4 shows the architecture of the Java application part of EHR Exchanger using exogenous connectors. The connectors in the Level 1 are unary invocation connectors and play the role of invocators for method calls to each component. The Level 2 and 3 connectors are pipe connectors, meaning that they connect to other connectors whatever they
are invocation, pipe or other types of connectors. In case of EHR Exchanger, all Level 2 pipe connectors compose two invocation connectors shown as Figure 5.4. The role of pipe connectors is to define the sequence of method calls for connected components. Namely, pipe connectors have the control and data flows, and pass the control to other connectors or allow invocation connectors invoke methods of a certain component. Particularly, based on the sequential flow, EHR system mostly uses pipe connectors and runs as we can see in the scenario for the prototype implementation in that we can implement by only using pipe connectors to compose other connectors.

```java
// create Level 1 connectors
invGenerator = new Invocation(generator1);
invMapper = new Invocation(mapper1);
invSender = new Invocation(sender);
invCA = new Invocation(ve);

// create Level 2 connectors
// p2
CDGenerator = new Invocation[2];
CDGenerator[0] = invMapper;
CDGenerator[1] = invGenerator;
p2 = new PipeForCDGenerator(CDGenerator);

// p3
CDSender = new Invocation[2];
CDSender[0] = invSender;
CDSender[1] = invCA;
p3 = new PipeForCDSender(CDSender);

// create Level 3 connectors
conCD = new Connector[2];
conCD[0] = p2;
conCD[1] = p3;
p1 = new PipeForCD(conCD);
```

Figure 5.5: The creation of connectors for EHR Exchanger.

Figure 5.5 shows how connectors are created according to the architecture. When the invocation connectors are initiated, each component is directly registered in the responsible invocation connectors. For example, one of the invocation connectors, `invGenerator`, registers the ‘generator’ component, which is invocated by only this connector. After that, the invocation connectors are registered in the responsible Level 2 pipe connectors, and the Level 2 pipe connectors are registered in the Level 1 pipe connector. To look into the implementation of Level 2 pipe connectors which compose two invocation connectors, Figure 5.6 describes the control of one of pipe connectors, p2, for CD generator. As the connector encapsulates the control and data flow, the execution method of the connector consists of its data and the execution of the lower level connectors. Shown as Figure 5.6, the String variable, `patient_id`, is a shared data between the mapper and generator components so that it shows how the data flow is defined. In addition, the execution method call of the invocation connectors, `invs`, controls the sequence of the method call. The other pipe connector, p2, also is implemented in the similar manner like the pipe connector, p1, is.
The Level 3 pipe connector composes these p1 and p2 connectors. On the contrary to the Level 2 connectors which connect with the invocation connectors, the Level 3 and higher level connectors can compose all types of connectors. Thus, the Level three connectors should register the generic connector implemented as a super class. Namely, all connectors must extend the super class, Connector defined by Lau et al. [42]. Figure 5.7 shows the Level 3 pipe connector extends the Super class, Connector, as well as two Connectors are register by the constructor.

```
public void execute(Method[] ms, Object[] params)
{
    Object tmp;

    // required data for calling methods
    String patient_id = (String) params[0];

    // sequencing of method calls
    // call mapping method of Mapper component
    // execute call
    invs[0].execute(ms[0], params);
    tmp = invs[0].getResult;

    //+++ call CDA generation method of Generator component
    // prepare arguments for this method
    Object[] tmp2 = new Object[2];
    tmp2[0] = patient_id;
    tmp2[1] = tmp;

    // execute call
    invs[1].execute(ms[1], tmp2);
    this.result = invs[1].getResult;
}
```

Figure 5.6: Implementation of the pipe connector, p1.

```
public class PipeForCD extends Connector{
    private Connector[] cons;
    public Object result;
    public PipeForCD(Connector[] is)
    {
        this.cons = is;
    }

    @Override
    public void execute(Method[] ms, Object[] params)
    {
        Object[] tmp;

        Method[] nForP2 = new Method[2];
        nForP2[0] = ms[0];
        nForP2[1] = ms[1];

        // sequencing of method calls
        cons[0].execute(nForP2, params);
    }
}
```

Figure 5.7: The Level 3 connector, p1.

Until now, we look into the implantation of connectors. After this step, we can use these connectors as composition operator. Thus, we should defined initiating operations for all component and invoke the execute method of the top level connector. By only calling this method, all operations are controlled within the all connectors and each computation of all components is executed according to the control of the all connectors. Generally, there are
other types of connectors such as selector connectors which select one connector for executing[42]. However, EHR Exchanger is focused on the data flow and its proper treatment. In other words, it is not required to use selector connectors because the control flow is somewhat simple as the scenario described in the section 5.1.2. Figure 5.8 finally shows the initial operations for each component and calling the top level connector, p1.

```java
// initiate operations for all components
Method[] mn = mapper1.getClass().getMethods();
Object[] margs = new Object[2];
margs[0] = patient_id;
margs[1] = all_values_from_patient;
Method[] gn = generator1.getClass().getMethods();
Method[] sn = sd.getClass().getMethods();
Method[] ws = ws.getClass().getMethods();

// assign methods for level 3 pipe connector
Method[] ns = new Method[4];
ns[0] = mn[0]; // doMapping
ns[1] = gn[0]; // getCOADocument
ns[2] = sn[1]; // SendEC
ns[3] = ws[2]; // wssend

// execute the methods in level 3 (top level) pipe connector
p1.execute(ns, margs);
```

Figure 5.8: Initial operations and invocation of the execution of the Level 3 pipe connector.

By using connectors, all components can be separated from other components as we can see the architectural hierarchy shown as Figure 5.4. The control and data flow are just implemented within connectors. Thus, any component does not invoke other components’ methods and as a result, the coupling between components can be reduced. The part of Java web application of EHR system also can be implemented by the same manner. However, to simplify the experimental study, we will compare the Java application parts between all prototypes.

### 5.1.5 Prototype 3: Implementation using mediator connectors.

Before implementing the prototype 3, we should implement the framework of mediator connector described in Figure 4.11 and 4.12. Each factory class contains its responsible object. For example, the ‘InteractionFactory’ class has many ‘Interaction’ objects. In our implementation, ‘MediatorConnector’ initiates all factories. Then, the ‘MediatorConnector’ parses the attachment which defines all components and interactions constructing the system. In this stage, ‘ComWrappers’ are created by ‘ComWrapperFactory’ and all components are wrapped by ‘ComWrappers’. Methods, parameters, and interactions consisting of methods and parameters are created by the same manner. After that, the ‘MediatorConnector’ is ready to run interactions.

Based on the framework, we can design the interaction diagram for EHR Exchanger. We can see four components in Figure 5.4 and those components are reused again in this prototype through the mediator connector. Figure 5.9 shows the interaction diagram with all four components: Mapper, Generator, Sender, and WSSender.
From this interaction diagram, the attachment for EHR system can be composed. Figure 5.10 shows the part of the attachment represented by the CIML instance document for building the mediator connector.

```xml
<xml version="1.0" encoding="utf-8"/>
<Cim xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.w3schools.com D:\BTH\Thesis\CIMLSchema3.xsd">
  <contains>
    <Component Identifier="mapper1" ClassName="se.bth.cs.thesis.master.cd.component:Mapper" />
    <Component Identifier="generator1" ClassName="se.bth.cs.thesis.master.cd.component:Generator" />
    <Component Identifier="sd" ClassName="se.bth.cs.master.thesis.cd.component:Sender" />
    <Component Identifier="ws" ClassName="se.bth.cs.thesis.master.cd.component:ws:WSender" />
  </contains>
  <Interaction name="generateAndSendCD">
    <OperationalCall ComponentID="mapper1" name="getPatientID">
      <Outpar parameter Identifier="patient_id" ClassName="java.lang.String" />
    </OperationalCall>
    <OperationalCall ComponentID="mapper1" name="getVectorForPatient">
      <Outpar parameter Identifier="all_values_for_patient" ClassName="java.lang.Vector" />
    </OperationalCall>
    <OperationalCall ComponentID="mapper1" name="doMapping">
      <Inpar parameter Identifier="patient_id" ClassName="java.lang.String" />
      <Inpar parameter Identifier="all_values_for_patient" ClassName="java.lang.Vector" />
      <Outpar parameter Identifier="doc" ClassName="nu.xom.Document" />
    </OperationalCall>
  </Interaction>
</xml>
```

Figure 5.10: CIML Instance document for EHR Exchanger

Figure 5.11 shows how the mediator connector is created and run. In contrast to exogenous connectors shown in Figure 5.5 and 5.8, a mediator connector does not need to create several connectors and to invoke all methods because they are defined in the CIML instance document. It makes the subject program simpler than using exogenous connectors.
By using the mediator connector, we can easily compose the loosely coupled components. All operational method calls are run by the mediator connector in that the mediator connector plays the role of the composition operator. This mechanism makes it possible to increase the component reuse in terms of providing the easy composition manner.

5.2 Evaluation methods for component and its model

To conduct the experiment with the prototypes we implement, we have chosen the following component evaluation approach, Reusable Components (RC), which is one of four approaches categorized by Ismail et al. [29]. According to Ismail et al., Product Line Component (PLC) is the most mature approach. However, our prototypes is focused on each component rather than core asset in the software product line, and RC approach is as good as PLC approach [29]. Moreover, our prototypes are implemented as object-oriented manner. Thus, we also use the conventional metrics to measure the reusability of OO software, in particular, to measure the coupling between components by using Chidamber and Kemerer’s metrics [48].

Our prototypes are limited as a certain application area, Exchanging EHRs. Thus, in the PLC approach proposed by Washizaki et al. [35], we use only Self-completeness of Component’s Return Value or Parameter (SCCr/p) metric relating with the external dependency criterion of components. Other criterions such as existence of meta-information, observability, customizability are not proper in this uniform component implementation.

To measure external dependency, we follow SCCr defined by Washizaki et al. [35]:

Definition 5.1 (SCCr: Self-Completeness of Component’s Return Value)

SCCr(c) is a percentage of business methods without any return value in all business methods implemented within a component c:

\[
SCCr(c) = \begin{cases} 
\frac{B_v(c)}{B(c)} & (B(c) > 0) \\
1 & (Otherwise)
\end{cases}
\]

Where:

- \(B_v(c)\): the number of business methods without return value in c
- \(B(c)\): the number of business methods in c

Definition 5.2 (SCCp: Self-Completeness of Component’s Parameter)

SCCp(c) is the percentage of business methods without any parameter in all business methods implemented within a component c:

\[
SCCp(c) = \begin{cases} 
\frac{B_p(c)}{B(c)} & (B(c) > 0) \\
1 & (Otherwise)
\end{cases}
\]

Where:

- \(B_p(c)\): the number of business methods without parameters in c
Confidence interval of SCCr is from 0.61 to 1.0, meaning that if SCCr value is between these intervals, the external dependency of a component is low and it leads to high portability of the component [35]. In similar way, SCCp also has confidence interval from 0.42 to 0.77 and this means the low degree of external dependencies [35]. Namely, the less parameters of the business methods show higher portability because of the low external dependency.

To measure coupling between components, we selected Coupling between object classes (CBO) metric which is proposed by Chidamber and Kemerer’s metrics [48] and is also conventionally used for measuring the static coupling between components in the recent research [18]. In this study, we are measuring the coupling between components rather than classes in that we count how many inter-component interactions in the system as Ismail et al. evaluate their components [29]. In other words, Ismail et al. defines the total CBO of components as follows [29]:

\[ T_{CBO} = \sum_{i=1}^{n} \text{Coup}(i) \]

Where:

- \( n \): the number of components in the system which includes a set of components \( C = \{C_1, C_2, ..., C_n\} \)
- \( \text{Coup}(i) \): the CBO of \( C_i \)

Additionally, we extended the assumption of Ismail et al. to measure accurately:
- \( T_{before} \): the total CBO of components before composition in a system
- \( T_{after} \): the total CBO of components after composition in a system in prototype 1
- \( T_{ec} \): the total CBO of components after composition in a system in prototype 2 using Exogenous connectors
- \( T_{mc} \): the total CBO of components after composition in a system in prototype 3 using Mediator connector

Additional source code to compose or use components in the subject system can be an important factor for component reusability. If programmers should give their efforts to compose components even though loose coupled components, reusability can not be guaranteed in terms of the difficult usage for composing components. Thus, additional lines of code (LOC) for using components in the subject system can be measured to decide how reusing component is simple or complex.

Finally, we also need to measure the performance of each prototype. This measurement leads us to confirm the additional overhead when using other component models. As we increase the number of records which is treated by each prototype, the performance is experimented in the statistical way.

### 5.3 Experiment plan

Based on the evaluation methods that we present in the previous section, three prototypes will be compared with each other. First of all, to measure external dependency, we use SCCr/p metric for all prototypes. From the result of this measurement, we can be notified portability of all components used in each prototype. Second, coupling between components is the most important character in assessing the reusability of components. Thus, we use CBO metric to measure total coupling between components, and the comparison of all prototype components in terms of the coupling before or after composition in each prototype shows the degree of coupling between components according to the prototype implantation.
using new component models. Third, the additional lines of code to compose component are counted to see whether the component composition is simple in the subject system. We also need to consider a performance impact when we use recently proposed component models. Namely, we test each prototype in terms of performance. Performance measurement will be conducted by increasing the data records processed and measure the elapsed time for processing the records to transfer EHRs.
CHAPTER 6: RESULTS

This chapter presents the result of the empirical study we plan in Chapter 5 as well as the analysis of the experiment. By this step, we can validate whether new component models using exogenous connectors and mediator connectors guarantee component reusability.

6.1 Measurement of external dependency

Washizaki et al. [35] propose measuring external dependency by yielding $\text{SCC}_{r/p}$ to decide which components is portable. In the case of $\text{SCC}_r$ which counts the business methods without a return value, Prototype 1 (JavaBeans) has the highest value followed by prototype 2 (Exogenous) and 3 (Mediator) shown as Table 6.1. The confidence interval is from 0.61 to 1. Thus, we can say Prototype 1 and 2 have high portability. Usually, as $\text{SCC}_r$ increases, portability also increases. Namely, Prototype 1 has the highest portability.

Table 6.1: Measurement of $\text{SCC}_r$ for each component

<table>
<thead>
<tr>
<th>Components</th>
<th>Measurement of $\text{SCC}_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JavaBeans</td>
</tr>
<tr>
<td>Mapper</td>
<td>0.67</td>
</tr>
<tr>
<td>Generator</td>
<td>1</td>
</tr>
<tr>
<td>Sender</td>
<td>1</td>
</tr>
<tr>
<td>WSSender</td>
<td>1</td>
</tr>
<tr>
<td>Average</td>
<td>0.92</td>
</tr>
</tbody>
</table>

However, the measurement of Prototype 2 and 3 should be calculated by a different way because all components in Prototype 2 and 3 do not have any external dependency between components, and they communicate with other components by only connectors. Table 6.2 describes the result values of $\text{SCC}_r$ considering connectors of Prototype 2 and 3. From this result, Prototype 2 and 3 have the highest portability.

Table 6.2: Measurement of $\text{SCC}_r$ considering connectors

<table>
<thead>
<tr>
<th>Components</th>
<th>Measurement of $\text{SCC}_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JavaBeans</td>
</tr>
<tr>
<td>Average</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Measurement of $\text{SCC}_p$ also provides the degree of portability by counting the business methods without parameters. The result values of $\text{SCC}_p$ is shown as Table 6.3. As the confidence interval is from 0.42 to 0.77, all prototypes have low portability in terms of parameters of business methods.

Table 6.3: Measurement of $\text{SCC}_p$ for each component

<table>
<thead>
<tr>
<th>Components</th>
<th>Measurement of $\text{SCC}_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JavaBeans</td>
</tr>
<tr>
<td>Mapper</td>
<td>0.33</td>
</tr>
<tr>
<td>Generator</td>
<td>0</td>
</tr>
<tr>
<td>Sender</td>
<td>0</td>
</tr>
<tr>
<td>WSSender</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>0.08</td>
</tr>
</tbody>
</table>
However, \( SCC_p \) values also should be calculated by considering connectors for Prototype 2 and 3. All business methods in exogenous connectors used in Prototype 2 have parameters for each business method, meaning that exogenous connectors have high external dependency. In contrast to exogenous connectors, mediator connector has a parameter when only running interactions and as a result, Prototype 3 has 0.5 as \( SCC_p \) value. The \( SCC_p \) value of Prototype 3 is within the confidence interval, meaning that the portability of Prototype 3 has higher than other prototypes.

<table>
<thead>
<tr>
<th>Components</th>
<th>Measurement of ( SCC_p ) for each component considering connectors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JavaBeans</td>
</tr>
<tr>
<td>Average</td>
<td>0.08</td>
</tr>
</tbody>
</table>

### 6.2 Measurement of Total coupling

Total CBO values of prototypes are described from Table 6.5 to 6.7. In Prototype 1 (JavaBeans), a method in WSSender component is invoked by Sender component and as a result, the number of inter-component interactions is one shown as Table 6.5. We also implemented Main Frame component to display a user GUI as an external and additional component. Prototype 1 has 3 inter-component interactions with Mapper, Generator and Sender components before composing. After composing, the CBO value of Main Frame has 4 because WSSender component has the relationship with Main Frame via Sender component. In case of Prototype 2 (Exogenous), the CBO value of Main Frame component has 10 which value is originated from the number of invocations of exogenous connectors. That is because all exogenous connectors should be initiated in Main Component. However, the CBO value of Prototype 3 (Mediator) has 4 which is from the number of invocation for a mediator connector. In contrast to exogenous connectors, one mediator connector can compose all components. Thus, the CBO value 4 means the invocation of methods for one mediator connector. When we consider Main Frame components, the lowest CBO value is shown in Prototype 3 followed by Prototype 1.

<table>
<thead>
<tr>
<th>Components</th>
<th>Measurement of T(_{CBO}) in Prototype 1 (JavaBeans)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( T_{before} )</td>
</tr>
<tr>
<td>Mapper</td>
<td>0</td>
</tr>
<tr>
<td>Generator</td>
<td>0</td>
</tr>
<tr>
<td>Sender</td>
<td>1</td>
</tr>
<tr>
<td>WSSender</td>
<td>0</td>
</tr>
<tr>
<td>Main Frame*</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
</tr>
</tbody>
</table>

* Main Frame is the User GUI component.

Actually, the composition of components to implement exchanging EHRs is for Mapper, Generator, Sender, and WSSender except for Main Frame. Thus, to acutely analyze for each component models, the total CBO values without Main Frame. As shown in Table 6.8, Total CBO value without Main Frame means that Prototype 2 and 3 implemented by connectors has no coupling between other components. If we can say the Prototype 2 and 3 without Main Frame component are a composite component or a sub system, the component model using a mediator connector to compose components has the best one between prototypes in terms of coupling, regardless of whether the sub system is a whole system or the part of a whole system. Prototype 2 using exogenous connectors has no coupling when the sub system means a whole system. However, when we consider the Main Frame
component, prototype 2 shows more coupling than Prototype 1. We will discuss this difference in the next chapter in detail.

Table 6.6: Measurement of $T_{CBO}$ in Prototype 2 (Exogenous Connector)

<table>
<thead>
<tr>
<th>Components</th>
<th>$T_{before}$</th>
<th>$T_{ec}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapper</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Generator</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sender</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WSSender</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Main Frame*</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

* Main Frame is the User GUI component composed regardless of the exogenous connectors

Table 6.7: Measurement of $T_{CBO}$ in Prototype 3 (Mediator Connector)

<table>
<thead>
<tr>
<th>Components</th>
<th>$T_{before}$</th>
<th>$T_{mc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapper</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Generator</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sender</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WSSender</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Main Frame*</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

* Main Frame is the User GUI component composed regardless or the mediator connector

Table 6.8: Comparison of $T_{CBO}$ considering with or without Main Frame component

<table>
<thead>
<tr>
<th>Components</th>
<th>$T_{after}$</th>
<th>$T_{ec}$</th>
<th>$T_{mc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapper</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Generator</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sender</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WSSender</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Main Frame (MF)</td>
<td>5</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>Total w/o MF</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>6</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

6.3 Measurement of additional lines of code

Table 6.10: Measurement of additional lines of code

<table>
<thead>
<tr>
<th>Code type</th>
<th>JavaBeans</th>
<th>Exogenous Connector</th>
<th>Mediator Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>3</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Operational</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Misc*</td>
<td>3</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td>41</td>
<td>6</td>
</tr>
</tbody>
</table>

* Codes for the value assignment or initialization

The additional lines of code (LOC) for composing components can be a critical criterion in terms of easy composition of components. Simply, if programmers need to add many lines of code to composite components, we can say the composition of components is complex but not simple. In this regard, Table 6.10 shows the meaningful result for each prototype. The LOC value of Prototype 3 (Mediator) is the lowest, meaning that the component model used in Prototype 3 provides the simplest way between other component models in prototyping. In
the case of Prototype 2 (Exogenous), LOC is the largest one, meaning that composing components is more complex than Prototype 3. Although Prototype 2 using exogenous connectors is efficient in terms of coupling between components, programmer’s efforts using exogenous connectors is bigger than Prototype 1 (JavaBeans) implemented in one of the current component models, JavaBeans.

### 6.4 Performance measurement

We measure the elapsed time for performance test of each prototype. By increasing the number of transferred EHRs, we test the prototypes several times (10–30) and measure the average elapsed time. Figure 6.1 shows the performance test results for all prototypes. As we can know this result, when the transferred records are 500, the time interval between Prototype 2 (Exogenous, the slowest) and Prototype 3 (Mediator, the fastest) is only about 5 seconds. The fastest performance is shown in Prototype3 and the lowest performance is on Prototype 1 (JavaBeans).

![Figure 6.1: The performance test results for prototypes](image)

To see the detail difference, Table 6.11 describes performance changes based on the prototype 1. After 100 records, there is an improvement on performance of Prototype 3 with 7–11% more when comparing with Prototype1, and 12–15% more when comparing with Prototype 2. This result clearly shows, when we use the component model using a mediator connector, there is a benefit in terms of performance for exchanging EHRs.

<table>
<thead>
<tr>
<th>The number of transferred EHRs</th>
<th>JavaBeans</th>
<th>Exogenous</th>
<th>Mediator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100%</td>
<td>92%</td>
<td>97%</td>
</tr>
<tr>
<td>10</td>
<td>100%</td>
<td>94%</td>
<td>102%</td>
</tr>
<tr>
<td>20</td>
<td>100%</td>
<td>99%</td>
<td>107%</td>
</tr>
<tr>
<td>50</td>
<td>100%</td>
<td>86%</td>
<td>99%</td>
</tr>
<tr>
<td>100</td>
<td>100%</td>
<td>96%</td>
<td>111%</td>
</tr>
<tr>
<td>200</td>
<td>100%</td>
<td>95%</td>
<td>107%</td>
</tr>
<tr>
<td>500</td>
<td>100%</td>
<td>95%</td>
<td>109%</td>
</tr>
</tbody>
</table>
6.5 Summary

From the experiment results, we can identify these following facts:

- New component models using exogenous and mediator connectors has less external dependency than the JavaBeans component model when considering components with connectors. (portability)
- The prototype using a mediator connector has no coupling between other components. (loose coupling)
- The prototype using a mediator connector provides programmers the simplest composing way among other prototypes. (easy composition)
- There is a performance improvement when using the mediator connector. (performance)

In this regard, the recently proposed component model using a mediator connector is higher reusability than other component models in prototyping. Thus, we can say that the approach of the component model using a mediator connector can be applied to increasing the reusability of the components for exchanging EHRs. In the next chapter, we will mention the additional interpretation of the experiment results.
CHAPTER 7: DISCUSSION

This chapter discusses the results of the empirical study in detail as well as the limitations our experiment and its evaluation. Moreover, based on the limitations, we figure out the future works to improve this study.

7.1 Detail Interpretation of results

7.1.1 External dependency

In the previous chapter, the SCCr value representing external dependency is shown as high in Prototype 2 (Exogenous) and 3 (Mediator) with the SCCr value 1 shown as Table 6.2. However, the SCCr value ‘0.92’ of Prototype 1 (JavaBeans) also has as high as other prototypes. The reason why the SCCr value of Prototype 1 is little bit lower than others is that Mapper component set member values using business method rather than write methods such as ‘set’ methods. If Mapper component uses the write methods to write values to member variables in the component, the SCCr value of Prototype 1 will be 1.0 as other prototypes. Using write methods is up to how programmers set values in what ways. Namely, according to programmers intend, external dependency can be improved. However, the most important thing is that the component models using exogenous and mediator connectors provide the framework for the lowest external dependency regardless of programmers’ intend.

7.1.2 Total coupling

Coupling between Objects (CBO) is one of factors to evaluate the component reusability [18]. Without considering Main Frame component, Sender component in Prototype 1 (JavaBeans) is the only component having coupling with WSSender component. That is because WSSender component is invoked by Sender component. Actually this invocation order is decided by programmers, meaning that if programmers design that WSSender component is not invoked by other components, Sender in Prototype will also be loosely coupled like other components. Depending on external dependency, this result also can be controlled by programmers’ intend.

Additionally, we found out an interesting result when we consider Main Frame component. As shown in Table 6.8, Prototype 2 (Exogenous) has more $T_{CBO}$ value than Prototype 1. If we regard the composite with Mapper, Generator, Sender, and WSSender components as a sub system, the EHR Exchanger is implemented by composing Main Frame and the sub system compose by four components. In all prototypes, the sub system is composed with Main Frame component by tight coupling. In other words, the component model using exogenous connectors should invoke all connectors in Main Frame component. This makes Prototype 2 tight coupling. In other words, the component model using exogenous connectors reduce the reusability of the sub system when it is composed other external components. In this case of composing with external components, for example Main Frame component, it is better not to use exogenous connectors as composition operators. In contrast to exogenous connectors, to make a sub system, only one mediator connector is required and coupling is only formed between one mediator connector and other components. This character of the component model using a mediator connector leads to the result shown...
in Table 6.8. This fact shows the mediator connector can be mixed with other component model with less coupling between components.

### 7.1.3 Additional lines of code

The result of additional LOC clearly shows that the component model using exogenous connectors is not proper as compositional operators in terms of its complexity to invoke methods. In the case of a mediator connector, the methods and arguments in the component are effectively manages by the attachment and Factory classes shown in Figure 4.12. However, exogenous connectors should add codes to register the components’ methods by programmers. Of course, the attachment for a mediator connector should be generated by programmers, but the mediator connector does not need to invoke each method in components because all methods’ invocation is automatically conducted by the definition in the attachment. Namely, the component model using a mediator connector provides the easy way to compose components as well as to manage components’ methods effectively by the attachment.

### 7.1.4 Performance

Table 6.11 represents how much Prototype 2 (Exogenous) is performed better than other prototypes. Over 100 EHRs, the performance of Prototype is more 12~15% than Prototype 1 (JavaBeans). To understand this performance difference, we need to look at Figure 7.1. In prototype 1 shown in Figure 7.1 (a), when Sender component is initiated, the component has a new instance. However, the component model using a mediator connector shown in Figure 7.1 (b) just invokes the run method which executes all operational methods internally without re-instantiating components.

![Image](image.png)

(a) Invocating the operational methods in Prototype 1 (JavaBeans)

(b) Invocating the operational methods in Prototype 3 (Mediator)

Figure 7.1: Comparison of the operational methods’ invocation between Prototype 1, 3

Of course, programmers can control the useless component instantiation by modifying source codes. However, we can know the component model using a mediator connector prevents unnecessary instantiating of components automatically. In other words, a mediator connector has a benefit in terms of decreasing impacts on performance.
7.2 Issues of the component model using a mediator connector

The experiment results show the component model using a mediator connector is a better approach comparing with other approaches using exogenous connectors and JavaBeans. In particular, the mediator connector plays the role of an effective compositional operator as this empirical study proves. However, the mediator connector has limitations when implementing systems. First, it is not possible to assign the initial parameters of the first operational method in the current mediator connector mechanism. Thus, each component should have read/write methods. The first two lines of Figure 7.2 (a) are the additional code setting the initial arguments before executing operational methods. Figure 7.2 (b) shows the definition of the corresponding read methods to write methods. If the mediator connector mechanism supports passing the initial values to the run method, it can be possible to run operational methods without additional read/write methods. The improvement of the mediator connector in terms of the initial arguments can be a future work.

```
// invoke write methods before run operational methods
sw.bhh.cs.thesis.master.cd.component.Mediator.setPatientID(patient_id);
sw.bhh.cs.thesis.master.cd.component.Mediator.setVectorForPatient(all_values_from_patient);

// result of execution of methods
mediatorCon.run("generateAndSendID");
Object result = mediatorCon.getResult();
```

(a) Invocation of the write methods

![Interaction](interaction.png)

(b) The part of the attachment to define the read methods

Figure 7.2: the use of read/write methods

Second, all methods and arguments are instantiated as objects, and components are wrapped by wrapper classes. The main strength of using factory classes shown in Figure 4.12 is the effective management of methods and arguments. However, it has a trade-off because all methods and arguments are object in the mediator connector so that huge heap memory size is required as the number of methods and arguments are increased.

7.3 Threat to validity

There are four major threats to the validity of this study:

1. **The JavaBeans component model can not be the representative component model.**

JavaBeans component model is good for comparing with the new component models using connectors. That is because all of prototypes are implemented by Java technologies. However, the JavaBeans model is just one of the current component models using in the software industry. As Lau and Wang [6] proposed the taxonomy of the current software component models, we need to use other component models.
such as EJB, COM, .NET, CCM, Web Services, Koala, KobrA, SOFA, Acme-like ADLs, UML 2.0, PECOS, and Fractal.

2. **The new component models that we experiment are only focusing on easy composition ways.**
   As we present in the chapter 4, there is another component model which calls Active Binding Technology. This component model is not implemented as one of prototypes. That is because the component models using connectors solve the problem focused on the easy composition way rather than matching different interfaces of components which is the main motivation of Active binding Technology. Namely, the experiment we conduct is just focused on the composition way by using connectors.

3. **Discharge letters might not be representative.**
   Discharge letters are one of the mostly-used clinical documents in hospitals. However, it does not mean discharge letters are the representative clinical documents. This can also limit our experiment result because EHR exchanger consisting of Mapper, Generator, Sender, and WSSender is not tested for other clinical documents. To strengthen the experiment results, we need to test with all other clinical documents. For the future work, experiments with various clinical documents should be conducted.

4. **The component models using connectors are not fully developed for the industrial use.**
   The component models using exogenous and mediator connectors are just proposed in the level of research but not practical software industry [6, 18, 45]. Thus, the prototypes we implanted are limited in the view of research. However, we believe the concept of reuse approach using mediator connector can be applied to practical EHR exchanger development.

5. **The environment for measuring performance does not consider network communication load since the performance is measured in a single system which plays the roles of both client and server at the same time.**
   EHRs are exchanged by the network, meaning that the speed of the network affects on performance for exchanging EHRs. However, the current experiment results were generated from the single system working for client as well as server. The limited environment can make the performance results biased. This also can be one of threats.

### 7.4 Future works

Based on these threats, we can come out future works as follows:

- **Empirical study by using various clinical environments**
  In our experiment, we used only discharge letter. However, discharge letters are just one of EHRs. If we can conduct the experiments using not only discharge letters but also other kinds of clinical documents. In particular, medical images such as Magnetic Resonance Imaging (MRI) and X-ray imaging also play an important role. However, adding medical images in this thesis requires other standards such as DICOM (Digital Imaging and Communication in Medicine) so that it takes same efforts as writing a new thesis in a different view. Thus, as DICOM standard allows a faster and more effective care to the patients, the experiments using medical images is also an important future work to be accomplished.
• The improvement of the mediator connector
The mediator connector has a limitation on setting the initial parameters originated from the static attachment. Moreover, the mediator connector requires more heap memory size even it leads better performance. Thus, there is a room for improvement for the mediator connector. The improvement of the mediator connector also can be a future work to contribute to exchanging EHRs.
SUMMARY

Health topic has been the critical issue for a long time. When Information Communication Technologies (ICTs) meet health problems, it thrives drastically National Health Information Infrastructure (NHII) which has been developed in several EU countries, the United States, Canada, and Australia. The efforts in constructing NHII contribute to improving healthcare services and health-related research studies for people. One of the challenges to construct NHII is Electronic Healthcare Records (EHRs). In particular, exchanging EHRs is the core functionality of NHII and used in many healthcare systems such as EHR systems between hospitals, medical research centers, and governments. In this regard, the reusability of the components for exchanging EHRs is the promising issue to be solved. By this study, we identify the better reuse approach in terms of software components. The validation of the reuse approach is conducted by the experiment using the prototype implementation based on recently proposed component models using exogenous and mediator connectors.

Before the empirical study, we have studied several EHR standards such as HL7 CDA, CEN ENV 13606, openEHR, and DICOM. For the prototyping to exchange EHRs, we chose one of EHR standards, HL7 CDA, which commonly used in that all contents in any clinical document can be expressed with CDA (Clinical Document Architecture). Using HL7 CDA to implement EHR exchanger, we answer these research questions:

1) What is the better component model to guarantee component reusability for exchanging electronic Health Records between JavaBeans and component models using exogenous connectors or mediator connectors?

We have studied three component models which are recently proposed and are called exogenous connectors, mediator connectors, and Active Binding Technologies (ABT). Exogenous and mediator connectors are focused on providing the easy composition way. In contrast to the component models using connectors, ABT is motivated by matching different interfaces to assemble components. Because of the different focuses, we implemented prototypes using exogenous and mediator connectors not ABT. These prototypes are compared with the prototype implemented by one of the commonly-used component models, JavaBeans.

2) How can component reusability be evaluated for systems implemented by using JavaBeans, exogenous connectors, and mediator connectors?

To evaluate each prototype, we measure external dependency (SCC\(r/p\)), Total coupling between objects (CBO), and additional lines of codes (LOCs). From this evaluation, we could know the mediator connector is a better approach than other component models in term of component reusability. Moreover, we found out the mediator connector is flexible to compose with the other component models with minimal coupling.

3) How big is a performance impact when each component models such as JavaBeans, exogenous connectors, and mediator connectors are adopted for constructing software systems?

Our major concern about using reusable components is performance. According to the experiment, using a mediator connector prevents unnecessary instantiating of components and as a result, it shows better performance than other component models used in prototyping.

In this study, the reuse approach using a mediator connector supports component reusability very well in terms of external dependency, total CBO, additional LOC, and better performance when comparing with other component models using exogenous connectors.
and JavaBeans. And the prototyping results will be a critical part of the blueprint for EHR Exchanger in constructing NHII. Moreover, scientific validation and analysis results by the empirical study provide the objective, academic basis for component reusability. Namely, we believe that the reuse approach using a mediator connector will be the better way for the implementation for exchanging EHRs. However, the experiments using more component models and various clinical documents are required in constructing NHII for the future works. In this regard, this study is a first step in this direction.
REFERENCES


