A Pattern Language for Class Responsibility Assignment for Business Applications

Soojin Park
Graduate School of Management of Technology
Sogang University
Seoul, Korea

***Abstract*—Assigning class responsibility is a design decision to be made early in the design phase in software development, which bridges requirements and an analysis model. In general, assigning class responsibility relies heavily on the expertise and experience of the developer, and it is often ad-hoc. Class responsibility assignment rules are hard to be uniformly defined across the various domains of systems. Thus, the existing work describes general stepwise guidelines without concrete methods, which imposes the limit in deriving an analysis model from requirements specification without any loss of information and providing sufficient quality of the analysis model. This study tried to grasp the commonality and variations in analyzing the business application domain. By narrowing the subject of the solution, the presented patterns can help identify and assign class responsibilities for a system belonging to the business application domain. The presented pattern language consists of six segmented patterns, including 19 variations of relationship type among conceptual classes. Each sequence of a use case specification could be analyzed as the result of weaving a set of the six segmented patterns. A case study with a payroll system is presented to prove the patterns' feasibility, explaining how the proposed patterns can develop an analysis model. The coverage of the proposing CRA patterns and enhancement of implementation code quality is discussed as the benefit.**

Keywords*—Class responsibility assignment; analysis pattern; business application; sequence diagram

I. Introduction

Developing an analysis model is the first phase in software development where abstract solutions are contrived. In the analysis model development, the task that is the most challenging and requires high creativity is assigning class responsibilities. Due to the nature of the task, responsibility assignment has heavily relied on the developer's experience and knowledge about the application domain. The class responsibility assignment (CRA) is hard to teach and apply [1]. On the other hand, it is hard to revise the wrong assignment of responsibilities to classes by adding other design patterns or architectural styles in successive phases.

The GRASP pattern [2] is remarkable and traditional among several approaches introduced to solve the CRA problem. However, it provides several fragmentary solutions and still requires lots of ad-hoc decision-makings to implement the patterns in a specific system. Since introducing the GRASP pattern, several approaches [3-5] that try to lessen the heuristic aspect of the CRA problem have been proposed. Nevertheless, their limitation is that they propose a way to evaluate the CRA results rather than assign responsibility itself. The posterior evaluation cannot reduce developers' efforts which are already exerted for CRA.

This study presents a pattern-based approach for assigning responsibility, which bridges analysis modeling and design modeling. CRA problems for business applications can eventually be decomposed into a set of CRUD operations on information: creating (C), reading (R), updating (U), and deleting (D). Thus, a data transaction is decomposed into six fragments and designed a CRA pattern for each fragment. This study also provides a way to compose the six fragmented CRA patterns for realizing a sequence diagram for each scenario in use case specifications. According to the given scenario, the sequence diagram can be composed of 2~6 CRA patterns. The links and messages that appeared in sequence diagrams are reflected as relationships between classes and responsibilities of each class. Each CRA pattern is represented by a uniformed template similar to the Gang of Four (GoF) pattern template [6] and composed of predefined variables and constants. The information developers extract from use case specifications is used to substitute variables in the CRA pattern and decide which patterns compose a complete sequence diagram for a scenario. In other words, developers can make an analysis model from the requirements model by mapping the information from use case specifications into each CRA pattern in developing an analysis model from the requirements model.

Compared with other related studies, the differentiated point of the proposed CRA pattern is as follows: the most assignment result of class responsibility is not a set of the tentative candidates but a final decision itself. The limit of the other existing work on class responsibility problems is that developers must select one among multiple candidate responsibilities or revise the candidates even after applying proposed methods. The reason is that most methods do not have a limit on the scope of their application. A solution proposed by the approach to solving the CRA problem of all domains cannot embed the properties for each domain. As a result, even if it is a solution that automatically supports class responsibility assignment, developers must tailor it to fit the characteristics of the domain after applying the methods. This study limits the proposed CRA pattern's application scope to the business application domain to substantially reduce those kinds of developers' efforts. Instead, by embedding the inherent features of the business application domain into the patterns, most of the responsibilities extracted from the CRA pattern

application are included in the final version of an analysis model without any revise.

A case study is conducted to adopt a payroll management system to show the feasibility of using the proposed CRA patterns in developing an analysis model. The coverage of the responsibilities extracted from the CRA patterns is measured to show the benefit of the proposed patterns.The result explains that a considerable portion of the responsibilities can be systematically extracted by applying the CRA patterns and included in the final version of the analysis model. And, the enhancement of the code quality derived from the analysis model constructed by applying the CRA patterns is also evaluated.

The rest of the paper is organized as follows: Section 2 presents related works on the class responsibility assignment problem. Section 3 gives an overview of the presented CRA pattern language, and Section 4 introduces the representation of each CRA pattern. Section 5 demonstrates a case study using a payroll management system, and section 6 shows the evaluation result. Section 7 concludes the paper with future work.

II. Related Work

Most of the analysis patterns [7-12] tended to focus on providing a way to identify classes that abstract domain knowledge. The main objective of design patterns [7] published up to now is to solve specific problems for successive implementation steps or enhance specific software quality. Contrary to this trend, [2] designated designing objects with responsibilities step as the heart of developing an object-oriented system and introducing the GRASP pattern. GRASP presents nine design principles as patterns: information expert, creator, low coupling, protected variations, indirection, polymorphism, high cohesion, pure fabrication, and controller. The GRASP pattern addresses fundamental, common questions and fundamental design issues on assigning class responsibilities. However, the questions defined by the GRASP are too general, and some principles are more fundamental than others. Their solutions are rather guidelines than patterns that define constants and variables of a model.

Since the introduction of the GRASP, several studies have been dealt with the CRA problem. Bowman et al. introduced a solution for the CRA problem, which is based on a multi-objective genetic algorithm (MOGA) and uses class coupling and cohesion measurement [3][13]. The MOGA takes as input a class diagram to be optimized and suggests possible improvements to it. They implemented a case study that showed that the multi-objective genetic algorithm could fix various artificially seeded assignment problems. However, the result of the MOGA application is limited to fixing the information included in classes. It does not help to construct sequence diagrams that explain dynamic behaviors based on the fixed responsibilities.

In [4], another metaheuristic algorithm for detecting wrong assigned responsibilities and making an optimized CRA is introduced. The proposed four different algorithms (simply genetic algorithm, hill-climbing, simulated annealing, and particle swarm optimization) use the same class coupling and cohesion metrics. They transformed the CRA problem into a search problem by encoding the problem and defining the fitness function. Like the MOGA, they chose a multi-objective approach, normalizing and combining three different coupling and cohesion measurements into a single aggregated fitness function and implemented a case study on the ATM Simulation domain model. Thus, their pros and cons are similar to MOGA's ones. Although they provide a way to evaluate already completed CRA results and enhance the quality of a design model, the contribution is limited to the conceptual model. Moreover, enhancement opportunities are given after the end of the developers' CRA step. Thus, it is hard to reduce the effort of developers on the responsibility assignment step itself.

Unlike the formerly described two approaches using some algorithm for detecting errors after the end of whole CRA steps, [5] proposed a technique to detect any error in every CRA step. For every step in CRA, the editor automatically detects bad smells of the current CRA and suggests refactored CRAs as alternatives. Designers can accept or reject the suggested CRAs. By repeating the steps of the responsibility assignment and refactoring, designers can construct the more appropriate CRA. This study's contribution is that they suggest formal representation for informal guidelines in GRASP and automatic detection rules for finding bad smells, which is violating the guidelines. However, like other approaches, the developers should create any CRA result before detecting the CRA errors and refactoring, and the application scope is limited on conceptual models.

Whereas the studies mentioned above mainly want to automatically detect errors as a follow-up to the developer's class responsibility identification results, the studies in [14-16] take an approach to automatically extract and present design elements from use case specification. [14] proposes an automated method that extracts domain classes from parsing use case specification using the Natural Language Processing (NLP) technique. [15], like [14], creates a parsed use case description (PUCD), an intermediate step product, from parsing the sentences of the use case specification, and then proposes candidates to construct a class model. The final decision to construct a class model remains to developers. [16] presents an automatic generation of a conceptual model from requirements written in a natural language, English, and proves the quality of the generated models against human works. However, their work has limited in that the coverage of the proposed method over all kinds of natural language is not comprehensive. And, the inherent ambiguity in the natural language occurs, hesitating the decision if a specific noun is an attribute, class, or association. Extracting a conceptual model from requirements needs an abstraction phase and heuristic insights for a specific domain area. Thus, it still requires experts' decisions on the details of the automatically generated conceptual model. Considering the effort of the experts' confirmation on the results, the benefit from the automatic generation of a conceptual model is skeptical. For this reason, this study does not include a conceptual model in the scope of automatically generated modeling artifacts. The conceptual model extracted by experts is used as input knowledge for automatic class responsibility assignments.

As mentioned above, although such NLP-based approaches have partial benefits, there is a limit to replacing the abstraction process that indicates the properties of the application domain, with only parsing the use case specification as a natural language sentence and interpreting it grammatically.

As the different approaches to solving class responsibility assignments, [17-18] attempt to determine responsibilities automatically between classes by using a class diagram as an input. [17] introduces a method that proposes an appropriate number of classes using three hierarchical agglomerative clustering algorithms and two criteria (aggregation metrics and CRA-Index) in the class diagram. And a comparison of the result applying their solution and the MOGA application result is presented. On the other hand, [18] presents a strategy for automatically generating a basic behavior schema from the static view represented by a class diagram. Through the analysis of the relationships between classes, the basic operations required for each class are identified. However, [17-18] has a limit in that the automatically generated operations are specified in too general terms, which requires the final decision of developers on every generated responsibility. Thus, the generated result can be utilized as a guide or reference to decide each responsibility of classes. Still, the automatically generated responsibility, i.e., operation, is hard to participate in an analysis model without a final update from developers.

Consequently, the existing studies on the CRA problem succeeded in enhancing the quality of a design model by providing various evaluation methods. However, most of the current work provides class responsibility assignments as a reference artifact or a candidate artifact for software developers. The result requires developers' final decision or update. It means that they still have a limit in relieving the developers of the CRA step's painful and challenging decision-making burden.

III. A Pattern Language for Class Responsibility Assignment for Business Applications

Most pattern-based approaches are subject to questions regarding the completeness of the patterns used. The answer to the question can be found in the definition of the business applications of [19]: "a business application is an application with structured logic and transaction-based database which supports simultaneous access by other applications." From the definition of a business application, an idea that a business flow can finally be disassembled into atomic CRUD(Create/Read/Update/Delete) operations can be captured. Most of the complex business services provided by the domain are readily broken down into a series of CRUD operations.

For example, Fig. 1 shows that the four different business flows from different systems can be decomposed to the identical combination of a read data pattern and an update data pattern. Besides the CRUD operations, interactions with the environment in which the system is driven are also needed to realize a business flow in the applications. In this study, the atomic collaboration between classes for realizing each CRUD operation and interaction with the environment is designed as each CRA pattern. The objective of the proposed CAR pattern language is to provide a way to build an analysis model by generating a sequence diagram of the system belonging to the business application domain combining the atomically designed CRA patterns. In this study, the syntax to utilize the CRA patterns is also provided. So, for this reason, the proposed set of the CRA patterns is named a "CRA pattern language." First, this section presents an overview of the CRA pattern language.

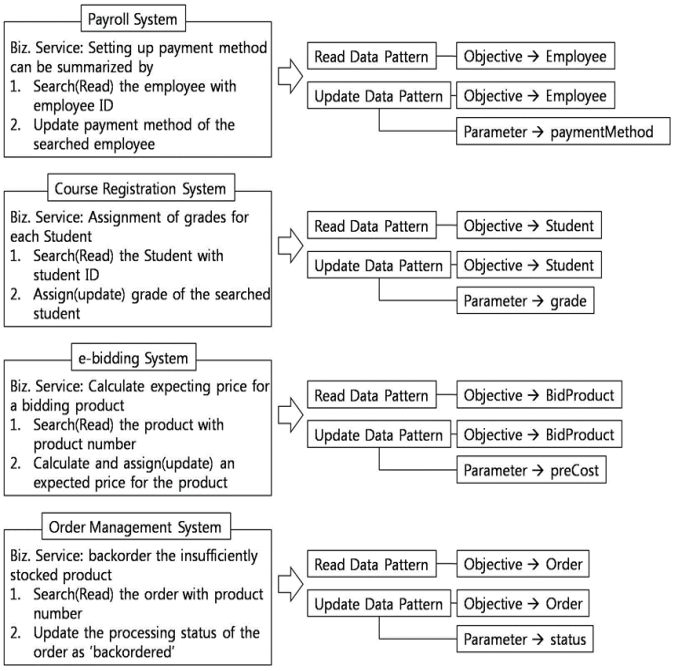


Fig. 1. Examples of Different Business Services Decomposed into the Same Atomic Collaboration Patterns.

A. Domain Model

A domain model for assigning class responsibilities for the business application contains all classes participating in the proposed pattern set. This study follows the Model-View-Controller (MVC) [20] pattern in identifying the role of participating classes in realizing a scenario of a use case specification. The participating classes that are divided into three analysis class stereotypes: << boundary >>, << control >> and << entity >>. These three stereotyped classes are arranged in separate packages that represent the basic three layers of business applications. The BizApplication package contains GUI form classes and <<boundary>> classes to interface with external systems. The BizProcess package contains <<control>> classes for managing flows in individual use cases. The BizLogic package contains <<entity>> classes to include actual business logic.

Fig. 2 depicts the whole class composing domain model for assigning class responsibilities of business applications. In Fig. 2, the question marks (?) in class names or operation names indicate pattern variables. The '+' mark, shown in class, operation, and parameter names, is an operator for concatenating two strings. Pattern variables are substituted with data values elicited from requirements documents to generate instantiated class or operation names during the pattern instantiation stage. For example, '?UCNm+ApprvalForm' is instantiated as RgstrCrsApprvalForm if the string value of

'?UCNm' is ‘RgstrCrs.’ Three classes in the BizApplication package are in charge of interfacing with external actors, including users and external systems. '?objective + ?DmType + Form' class is defined for user interfaces, and '?UCNm+AppvlForm' requests a specific approval from a supervisor role. '?Interface System' class is a boundary class for interfacing with other related systems. All of these three classes in the BizApplication package communicated with the control class, '?UCNm+Cntrl' in the BizProcess package, managing sequences of a flow. All messages from UI classes ('?UCNm+AppvlForm', '?objective+?DMType+Form') are blocked by '?UCNm+Cntrl' and all messages to the other system(s) go through the class. Except for the role of a proxy, the '?UCNm+Cntrl' class is responsible for calling appropriate messages to entity classes in the BizLogic package according to incoming requests.

The entity classes receiving messages from ‘?UCNm+Cntrl’ are '?objective', '?AssociatingClass', '?AssociatedAttribute Class' and '?DependentClass'. ‘?objective+Container’ classes are mainly generated by adopting one of the Read Data patterns. It plays as a container for some entity classes when it is needed to display multi-row data. ‘?objective+Transaction’ class is for only Transfer Data to Another System patterns. The role of the class is to specify data transactions by adding extra data (source system, destination system, length information, etc.) to the ‘?objective’ class. The information of this class is passed when the system should propagate the data manipulation results to other external systems. In Fig. 2, attributes of each class are suppressed to highlight the core of the patterns to assign each responsibility to the proper classes.

B. *Idioms of Pattern Language for CRA Problems in Business Applications*

In explaining an overview of the pattern language, this study follows the way of [21]. The pattern language application graph depicted in Fig. 3 shows how CRA patterns are applied during the modeling of applications. Still, it is not intended to show how the resulting system works, i.e., it is not a flowchart. In the original notation of [21], the main language patterns are split into mandatory patterns and optional patterns in the original notation. And, one entry point and several exit points are denoted to represent the pattern application flow.

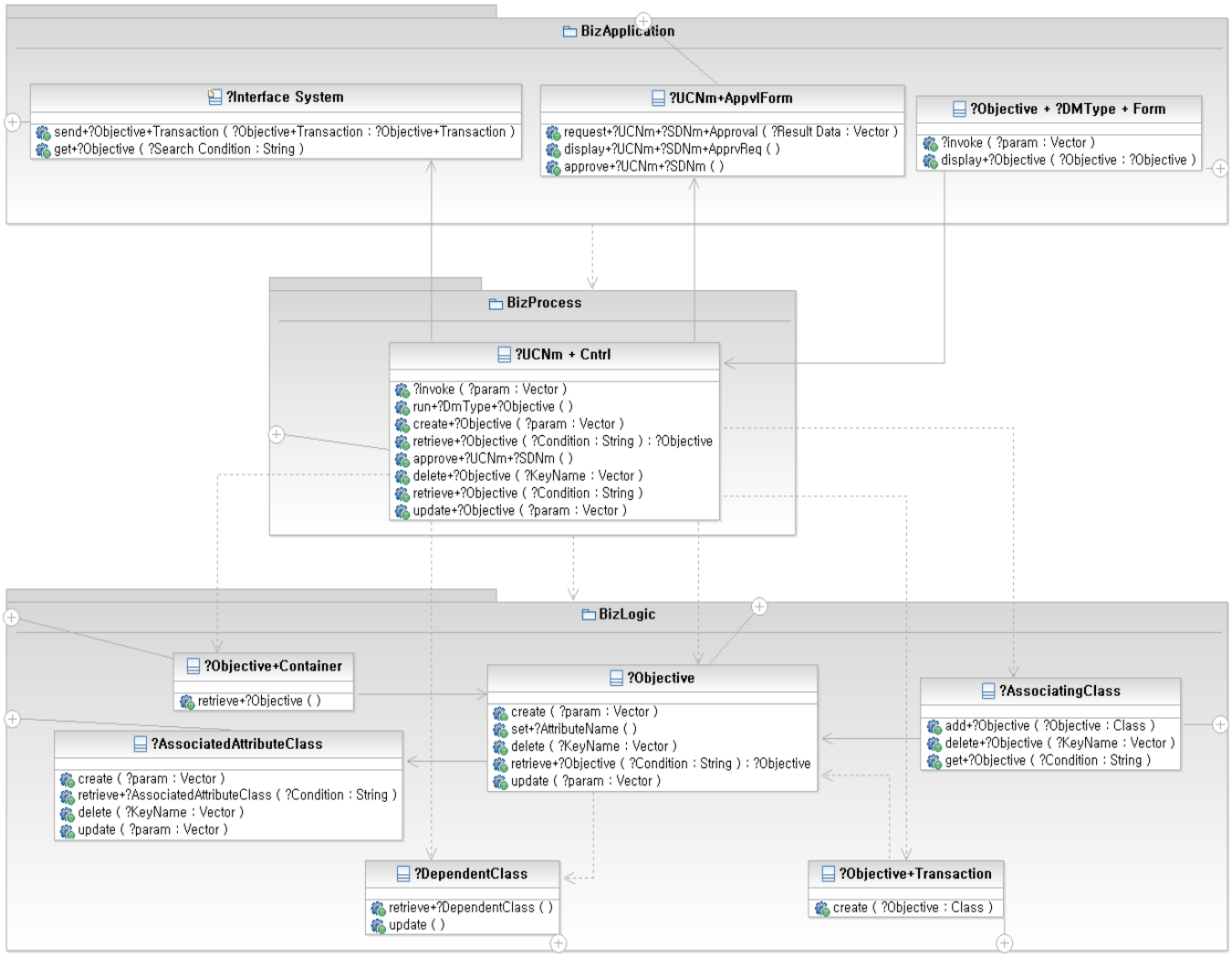


Fig. 2. Class Diagram for Domain Model of a Pattern Language for Class Responsibility Assignment.

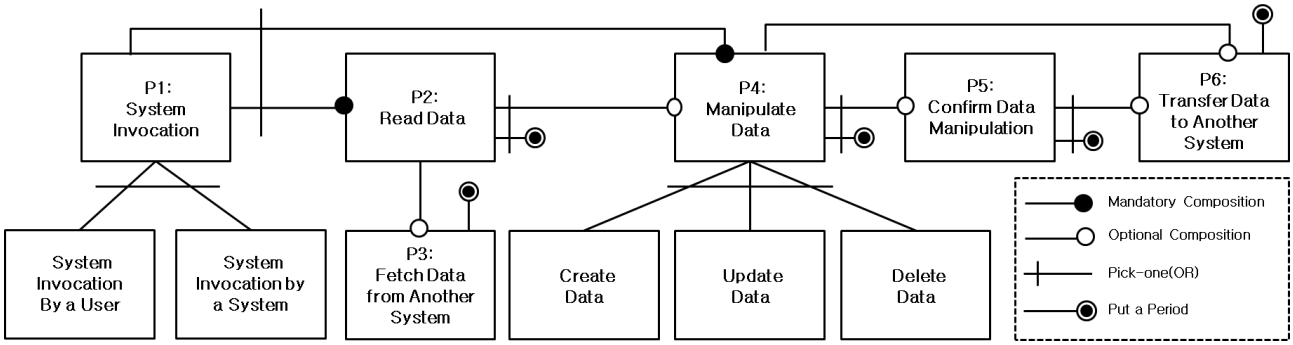


Fig. 3. Pattern Language Application Graph.

The firstly applicable pattern in order is the System Invocation pattern which gives a solution on how the system can be invoked. After applying the System Invocation pattern, the Read Data pattern can be successively applied as users usually check data before changing it. If the data is located in other systems, Fetch Data from Another System pattern can be applied. To be composed as a design fragment for reading some data, the possible pattern sequence is "System Invocation → Read Data" or "System Invocation → Read Data → Fetch Data from Another System." If the data source to be read is the system itself, the first pattern sequence will be applied. Otherwise, the second option should be applied. So, a typical pattern application sequence is "System Invocation → Read Data → Manipulate Data" patterns. But users could manipulate data without reading anything in some cases, which is also a typical sequence. Thus, the Read Data pattern can be an optional pattern like Fetch Data from Another System pattern.

After applying a pattern belongings to the data manipulation pattern, Confirm Data Manipulation pattern could be optionally applied if it requires any specific actions to reflect the result of data manipulation on the system. The other applicable pattern is Transfer Data to Another System pattern, applied when the data manipulation result should be reflected to another related system.

To sum up, the proposed pattern language is composed of 2 required patterns (System Invocation, Manipulate Data) and four optional patterns (Read Data, Fetch Data from Another System(s), Confirm Data Manipulation, Transfer data to another system(s)). The Read Data pattern is an optional pattern when it is applied with other data manipulation patterns. However, it can be a mandatory pattern when it is used to implement a scenario to show some information to users without any change on data. The minimum number of the patterns composing a scenario is two as the shortest sequence is "System Invocation → Read Data" or "System Invocation → Manipulate (Create/Update/Delete) Data." The maximum number of the applied patterns for realizing a scenario is six as the most extended pattern sequence is "System Invocation → Read Data → Fetch Data from Another System → Manipulate (Create/Update/Delete) → Confirm Data Manipulation → Transfer data to another system."

C. Process of Building an Analysis Model using CRA Pattern Language

With the CRA patterns, a sequence diagram to identify class responsibility from a scenario in a use case specification can be composed through three phases – use case analysis, CRA pattern weaving, CRA pattern instantiation. Fig. 4 shows each step of constructing an analysis model using the proposed CRA patterns, and the detail of each step is the following.

Use Case Analysis: (a) the reference artifacts are use case model and initially identified conceptual key classes. (b) The process starts with analyzing input use cases to identify the necessary information to populate CRA patterns through questions and answers. (c) A set of predefined questions is presented to the developer to decide patterns to be applied and elicit pattern variables to instantiate the chosen patterns.

CRA Pattern Weaving: Use case analysis results in a set of CRA patterns chosen to apply. Six patterns are presented in this study. Each selected pattern defines a segmented collaboration among participating classes, and the number of the selected patterns for a scenario is between two and six. (d) To realize a given scenario as an analysis model, the patterns to compose a complete sequence diagram should be weaved into a sequence diagram. The identically appeared lifeline between two CRA patterns becomes the connection point of the two patterns. From the P1 pattern to the P6 pattern, the required patterns are weaved step by step. At the end of CRA pattern weaving, we can get a skeleton of a complete sequence diagram for the target scenario.

CRA Pattern Instantiation: The skeleton of a sequence diagram resulting from the CRA pattern weaving step still has uninstantiated pattern variables. The value for each pattern variable is extracted in the previous use case analysis step. (e) From the answers to the questions, the values for pattern variables can be extracted. The instantiation of the composed pattern results in an analysis model with responsibilities of CRUD operations and other supporting operations for each analysis class. All of the responsibilities that appeared in the sequence diagram are registered as the operations of the key classes. Besides adding the operation to the existing key classes, new classes are also defined by applying CRA patterns.

The details of each phase will be explained with a tangible application case of a payroll management system in Section 5.

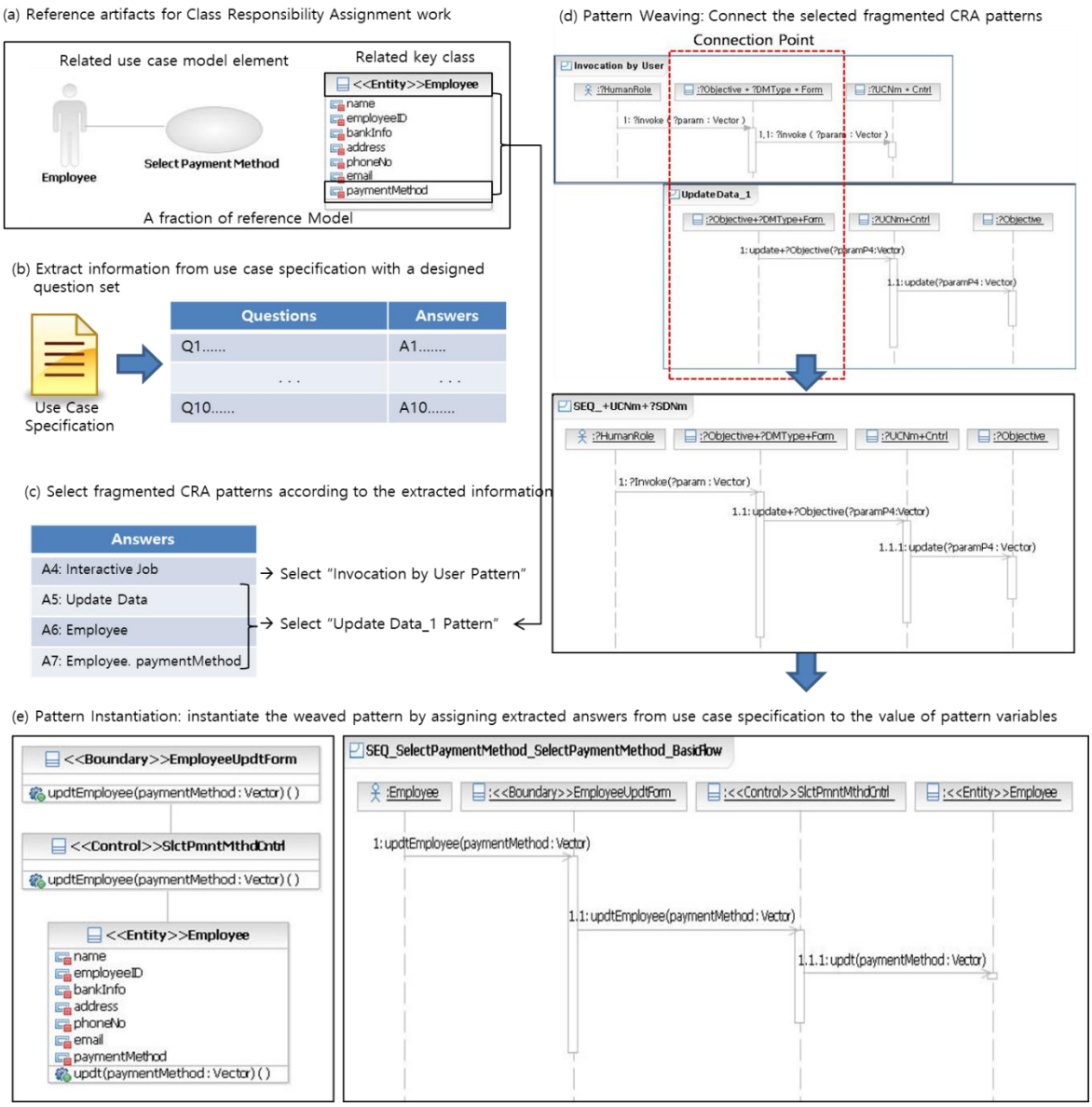


Fig. 4. Building an Analysis Model using CRA Pattern Language.

IV. Representation of a CRA Pattern

The GoF pattern template [6] is utilized to represent each CRA pattern. However, all of the compartments of the GoF pattern template are not used. Also Known As, Motivation, Known Uses, and Sample Code sections are not used as they are out of the concern of the CRA pattern. The followings are the sections and their brief descriptions of the CRA pattern.

1) Problem: The question to be solved with the pattern regarding the class responsibility assignment aspect.

2) Forces: The conditions be satisfied by applying the pattern.

3) Solution

a) Structure: The static view of the newly defined classes or their properties (operations and relationships), which participate in the interactions in the pattern.

b) Participants: The specification of roles of the classes participating in the interactions in the pattern.

c) Interaction: The dynamic view showing the collaboration among the classes specified in section 3.2 Participants. More than one interaction could be defined according to the relationship format of the <<Target>> role class and other classes.

4) Consequences: The guaranteed benefit from the application of the pattern.

5) Following patterns: Another CRA pattern connected to the next to build a complete sequence diagram.

6) Example: Simple application example is presented. The finally generated sequence diagram and the corresponding class diagram are provided for understanding the pattern.

Fig. 5 shows the specification of the "Create Data" pattern documented according to the template above. The "Create Data"

pattern has three different interactions, and each sequence diagram defines the collaboration among the participating classes according to the given condition.

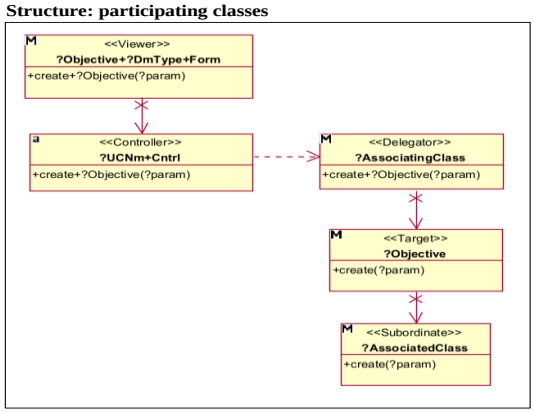
As specified in Fig. 5, the P4 patterns (Create/ Read/ Update/ Delete Data patterns) define several interaction variants according to class relationships and attributes. Table I lists up all interactions embedded in each CRA pattern. There are a total of 19 interactions that can be used to construct a sequence diagram, as shown in Table I.

Create Data Pattern

1. Problem
Who should be responsible for creating a new instance of some classes?

2. Forces
- Input data from a user should be created as an instance of a class.
- Responsibility assignment with high cohesion and low coupling should be accomplished.

3. Solution
3.1 Structure: participating classes

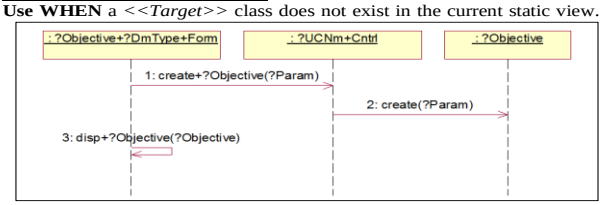


3.2 Participants

Role	Description
Viewer	A UI (User Interface) form class for accepting data required to be newly created.
Controller	A control class for conducting collaboration among classes for the realization of a given business flow.
Target	A newly created entity class as the result of the given business flow.
Delegator	An entity class including the <<*Target*>> class as a data member.
Subordinate	An entity class defined as a data member of <<*Target*>>.

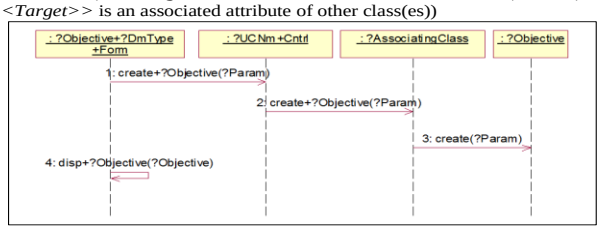
3.3 Interaction

Create Data 1: Simple Creation
Use WHEN a <<*Target*>> class does not exist in the current static view.



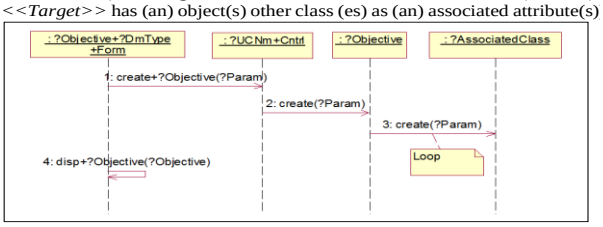
1. Input data from a *viewer* is passed to a *controller*
2. The *controller* creates a *target* object.

Create Data 2: Creation through an Associating Class
Use WHEN (a <<*Target*>> class exists in the current static view) AND (a <<*Target*>> is an associated attribute of other class(es))



1. Input data from a viewer is passed to a controller
2. The controller delegates create( ) responsibility to a delegator of the target object.
3. The delegator creates a target object.

Create Data 3: Successive creation of Associated Classes
Use WHEN (a <<*Target*>> class exists in the current static view) AND (a <<*Target*>> has (an) object(s) other class (es) as (an) associated attribute(s)



1. Input data from a viewer is passed to a controller
2. The controller creates a target object.
3. The controller delegates creation of subordinate to the target object.
4. The target creates subordinates object(s) as many as defined.

4. Consequences
A fragment of sequence diagram which is instantiated by the pattern conforms to the guidelines of the GRASP pattern. Thus, the instantiated design model can guarantee high cohesion and low coupling.

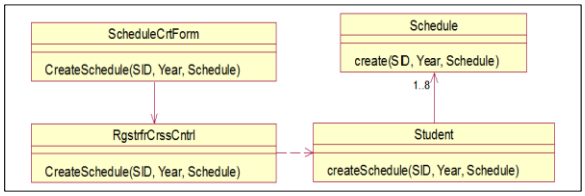
5. Following Patterns
Confirm Data Manipulation, Transfer Data to Another System

6. Example
- The flow of Event: Create a Schedule

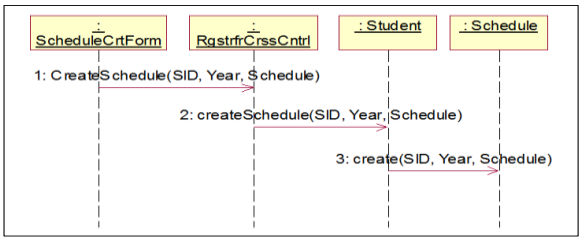
In the given conceptual model of a course registration system, Student class is associated with Schedule class. To compose a sequence diagram for the "creation of a schedule" flow, *Create Data 3* is selected and applied. The values elicited from requirements are as the following:

Pattern Variable	Input Value
?Objective	"Schedule"
?AssociatingClass	"Student"
?UCNm	"RgstrfrCrss" (abbreviation of RegisterforCourses)
?DMType	"Crt"
?paramP4	"SID, Year, Semester"

The design model fragment resulting from applying to *Create Data 3* is depicted in the following. The *Student* that is instantiated from ?AssociatingClass (Delegator) is the owner of *createSchedule( )* responsibility and creates an instance of the Schedule which is instantiated from ?Objective(Target) class.



Instantiated Class Diagram for "Create a Schedule" Flow



Instantiated Sequence Diagram for "Create a Schedule" Flow

Fig. 5. CRA Pattern Specification: Create Data Pattern.

TABLE I. THE LIST OF THE CRA PATTERNS AND THEIR INTERACTIONS

Pattern	Interaction	Applicability
System Invocation (P1)	Invocation by User	Use when a user invokes a flow of events
	Invocation by System	Use when a software system periodically accomplishes a flow of events
Read Data (P2)	Read_Data_1	Use when all retrieved data items are attributes of ?objective class
	Read_Data_2	Use when(Retrieved data item(s) is(are) distributed into more than two classes) AND (there exist association relationships among the classes)
	Read_Data_3	Use when retrieved data item(s) which is(are) not attributes of ?objective class belong to the class(es) that has(have) no association relationship with ?objective class
	Read_Data_4	Use when retrieved data item(s) which is(are) not attributes of ?objective class belong to ?AssociatedClass class and the others belong to the class(es) that has(have) no association relationship with ?objective class
	Read_Data_5	Use when an ?objective class is an associated attribute of other class(es)
Transfer Data from Another System (P3)		Use when the reading data from another interface system is needed
Create Data (P4)	Create_Data_1	Use when an ?objective class does not exist in the current static view
	Create_Data_2	Use when (an ?objective class exists in the current static view) AND (an ?objective is an associated attribute of other class(es))
	Create_Data_3	Use when (an ?objective class exists in the current static view) AND (?objective has (an) object(s) other class (es) as (an) associated attribute(s))
Update Data (P4)	Update_Data_1	Use when (an ?objective class does not exist in the current static view) OR ((an ?objective class exists in the current static view) AND (it is not the applicability of Update Data Pattern_2 and Update Data Pattern_3))
	Update_Data_2	Use when updated data item(s) that is(are) not an attribute(s) of ?objective class belong to ?AssociatedClass class
	Update_Data_3	Use when updated data item(s) which is(are) not an attribute(s) of ?objective class belong to the class(es) that has(have) no association relationship with ?objective class
Delete Data (P4)	Delete_Data_1	Use when (an ?objective class does not exist in the current static view) OR ((an ?objective class exists in the current static view) AND (it is not the applicability of Delete Data Pattern_2 and Delete Data Pattern_3))
	Delete_Data_2	Use when an ?objective class is an associated attribute of other class(es)
	Delete_Data_3	Use when (an ?objective class has another class as an associated attribute) AND (the relationship between the classes is an aggregation by value)
Confirm Data Manipulation(P5)		Use when an acquisition of higher leveled user for the reflection of data status changes is needed
Transfer Data to Another System(P6)		Use when the transformation of a data manipulation results to other system is needed

V. CASE STUDY: BUILDING AN ANALYSIS MODEL USING CRA PATTERN LANGUAGE FOR A PAYROLL MANAGEMENT SYSTEM

The proposed CRA patterns impose many constants and parameters(variables) in participating classes' attributes and responsibilities, confusing unintimate readers. So, this paper will explain the details of each phase of adopting CRA patterns in building an analysis model from a scenario of a use case with a specific system, a payroll management system, rather than discuss with a set of general constants and variables. The chosen scenario of the payroll management system is the basic flow of "Select a payment method," as shown in Fig. 6.

A. Use Case Analysis

First, a set of generic questions is presented for applying the proposed CRA patterns, as shown in Table II. The developer answers the questions based on use case specifications and already defined key classes. Each question is used for the developer to select a set of appropriate fragmented patterns and identify parameter values in each CRA pattern.

- Q1~Q2: Questions for extracting the name of the target use case and flow of events.

This question is for composing the name of the sequence diagram from a flow of events in a use case specification. The blanks in the answer strings to Q1 and Q2 are excluded when used as values for pattern variables, '?UseCaseNm' and '?SeqNm'. For example, if the answer to Q1 is "Select Payment Method" and the answer to Q2 is "Basic Flow," the values for '?UseCaseNm' and '?SeqNm' are "SelectPaymentMethod" and"BasicFlow," respectively. Thus, the newly constructed sequence diagram name is "SelectPaymentMethod BasicFlow."

The value of variables, '?UCNm' and '?SDNm' are decided by excluding vowels from '?UseCaseNm' and '?SeqNm'. The values for '?UCNm' and '?SDNm' are used to name classes and operations.

- Q3: Question for selecting a type of the *System Invocation* patterns.

The available answers to the question Q3 are "Interactive Job" or "Batch Job." If the answer for Q3 is "Interactive Job," System Invocation by a user behavior is selected as the second segmented collaboration pattern. In that case, additional question Q3.1 is given to designate the active actor of the target flow of event. The answer to Q3.1 is denoted as an actor, as

described in Table II. If the answer to Q3 is "Batch Job," System Invocation by a system behavior is selected, which describes a kind of automatic invocation of the target system according to predefined schedules. As it does not require an active actor, Q3.1 is skipped in this case.

The "Select payment method" flow is a kind of interactive job. Thus, the answer to Q3 is "Interactive Job." According to the use case specification, the flow starts with an event from the Employee. The answer to Q3.1 is "Employee," and it will be mapped to the active actor denoted as the pattern variable, '?humanRole' of the sequence diagram.

- Q4~Q6: Questions to select proper *Manipulate Data* pattern and extract required values for pattern variables.

To realize a flow as a sequence diagram, not all of the CRUD operations are used. Depending on the event description of a use case, a different set of CRUD patterns are used. Questions Q4~Q6 are designed for developers to help determine the CRUD operations set and extract values of pattern variables.

Use Case 5 : Select Payment Method

1. Brief Description

This use case allows an Employee to select a payment method. The payment method controls how the Employee will be paid. The Employee may choose to either: pick up his check directly, receive it in the mail, or have it deposited directly into a specified bank account.

2. Flow of Events

2.1 Basic Flow

This use case starts when the Employee wishes to select a payment method.

1. The system requests that the Employee specify the payment method he would like (either : "pick up", "mail", or "direct deposit").
2. The Employee selects the desired payment method.
3. If the Employee selects the "pick-up" payment method, no additional information is required.
   If the Employee selects the "mail" payment method, the system requests that the Employee specify the address that the paycheck will be mailed to.
   If the Employee selects the "direct deposit" method, the system requests that the Employee specify the bank name and account number.
4. Once the Employee provides the requested information, the system updates the Employee information to reflect the chosen payment method.

Fig. 6. Use Case Specification of "Select Payment Method."

TABLE II. QUESTIONS AND ANSWERS FOR "SELECT PAYMENT METHOD" FLOW ANALYSIS

No	Generic Questions	Answer	Pattern Variable	Selected Pattern
FLOW LEVEL				
Q1	What is the Use Case Name?	"Select Payment Method"	?useCaseName	N/A
		"SlctPmntMthd"	?UCNm	
Q2	What is the name of the flow of events?	"BasicFlow"	?seqNm	N/A
		"SlctPmntMthdBsc"	?SDNm	
Q3	What is the job characteristic?	"Interactive Job"	invocationType	P1: Invocation by a User
Q3.1	If it is an interactive job, what is the name of the active actor?	"*Employee*"	?humanRole	N/A
DATA MANIPULATION LEVEL				
Q4	What is the data manipulation type? (select 1 among creation/read/update/deletion)	"Update Data"(Updt)	?DMType	P4: Update Data
Q5	What is the objective data of the data manipulation?	"Employee"	?objective	
Q6	What are the data items to be changed after this update?	"*paymentMethod"*	?paramP4	
Q7	Is there any other data for an *Employee* to retrieve for the update of the *paymentMethod*?	"No"	needData	P2, P3 is not selected
Q8	Is the retrieved data located on another system? If it is, what is the system?	"No"	?interfaceSystem	P3 is not selected
Q9	To accomplish this flow of events, is it need to take any approval from someone?	"No"	NeedApproval	P5 is not selected
Q10	After completing this data manipulation, should the changed data be transformed to another system(s)?	"No"	NeedAnotherSystem	P6 is not selected

Q4 asks which data manipulation is needed to realize the target flow among creating/updating/deleting data. The abbreviation (Crt/Rd/Updt/Dlt) of the answer to Q4 is mapped to the value of the variable, '?DmType,' included in CRA patterns. After '?DmType' is designated, the next question, Q5, asks the objective of the designated data manipulation. The objective data of CRUD manipulation should be one of the key classes already given as an input artifact for building an analysis model. The next question, Q6, is applied only if the answer to Q4 is "Update Data." However, it does not mean that Q6 is differently designed according to the answer to Q4, in other words, the type of data manipulation. Q6 is designed to ask the property of the selected data manipulation. If the selected data manipulation type is "Delete Data," Q6 asks which property of the target class should be deleted. Therefore, while the answer of Q5 is one of the given key classes, the answer of Q6 should be one of the attributes in the selected class as the answer to Q5.

For example, in the case of the "Select payment method" flow of events, after all, the flow changes the value of the 'payment Method,' of the key class, 'Employee.' So, it is a kind of update manipulation. Thus, "Update Data" is the answer to Q4, and the abbreviation, 'updt,' is mapped as the value of '?DmType'. The objective data is the class, 'Employee.' The updated attribute is 'paymentMethod' as the answer to Q6, and it is mapped to the value of '?paramP4.'

- Q7: Questions to select *Read Data* pattern or not.

As depicted in Table II, for analyzing the "Select payment method" scenario, Q7 asks if an additional "Read Data(P2)" pattern is needed before the "Manipulate (Create/Update/Delete) Data" pattern. According to the given flow of events, the answer to Q7 is "No" as s a user selects his preferred payment method without retrieving any additional data from the system. Consequently, the "Read Data(P2)" pattern is not selected to compose a sequence diagram.

However, in the case that require additional retrieving data before creating/updating/deleting data or the case that main flow is for retrieving data (answer to Q4 is "Read Data"), answering the additional questions Q7.1 and Q7.2 are needed for extracting data for the "Read Data" pattern. The additional questions Q7.1 and Q7.2 are specified in Table III. Those questions ask the name of retrieving data and the retrieval conditions.

- Q8: Questions to select *Fetch Data from Another System* pattern or not.

If the answer to Q7 is "Yes," the answer to Q8 is required. In applying Read Data(P2) pattern to compose a sequence diagram, one of the checkpoints is the location of the data to be retrieved. Suppose the data location is not the target system, message.

Sequences to request the data to the system that is the source of the retrieved data. The required collaboration with the other system is defined in Fetch Data from Another System(P3) pattern. Thus, in this case, the P3 pattern should be weaved with the already selected P2 pattern. For the given example scenario, selecting the P3 pattern is not considered because it is not required to retrieve other data to update the payment method as the data resource is a user.

TABLE III. SUPPLEMENTAL QUESTIONS NOT APPLIED TO "SELECT PAYMENT METHOD" FLOW ANALYSIS

No	Generic Questions	Pattern Variable
Q7	Is there any other data for an '*?humanRole'* to retrieve to update the *'?objective'*?	?DmType
Q7.1	What is the name of the retrieving data?	?objective
Q7.2	What is the search condition for the retrieval of '*?objective'*?	?Condition
Q7.3	What are the retrieving attributes?	N/A
Q8	Is the retrieved data located on another system? If it is, what is the system?	?interface System
Q8.1	What data should be transferred from *?InterfaceSystem*?	?objective
Q8.2	What is the search condition for the retrieval of ?objective?	?condition
Q9	To accomplish this flow of events, is it need to take any approval from someone?	NeedApproval
Q9.1	Who is responsible for the data confirmation?	?actorNm
Q10	After completing this data manipulation, should the changed data be transformed to another system(s)?	NeedAnotherSystem
Q10.1	What is the destination system of the data transfer?	?interface System
Q10.2	What is the additional data to be transferred except ?objective data?	?addData

- Q9: Questions to select *Confirm Data Manipulation* pattern or not.

If the answer Q4 is one of the "create/ update/ delete data," the Manipulate Data(P4) pattern is selected, question Q9 should be considered. Q9 asks if any approval is needed to save data manipulation results or not. As the given example, if it is required to get a confirmation from any actor after updating the payment method of an employee, the answer to Q9 should be "Yes," and Confirm Data Manipulation(P5) pattern will be selected. P5 pattern defines the message sequences to request confirmation to an actor responsible for the approval of the data manipulation and to approve it into the target system.

In the case of payment method update, however, it is not required any other confirmation to select the payment method of own Employee. So, the answer to Q9 is "No," and the P5 pattern will not be selected.

- Q10: Questions to select *Transfer Data to Another System* pattern or not.

The other question to be considered when Manipulate Data(P4) pattern has been selected is Q10. In some cases, changes in data in the target system should be reflected in another system. The change should be propagated to the data source system when the changed data source is not the target system but the other system. In that case, the data source system should already have been identified as a passive actor in the given use case model. The transformation of the changed data to the passive actor is defined in Transfer Data to Another System(P6) pattern. Q10 asks if the changed data should be

transformed to another system after the completion of data manipulation. The P6 pattern will be selected and weaved with the P4 pattern to compose a sequence diagram when the answer to Q10 is "Yes."

In the given example case, the changed data, "payment Method" is an attribute of the class, "Employee," saved in the target system itself. The answer to Q10 is "No." Consequently, the P6 pattern will not be selected.

B. CRA Pattern Weaving

A set of CRA patterns necessary to implement the given flow is selected from among the six segmented CRA patterns by analyzing the given flow of the use case specification and answering each predefined generic question introduced above. Most CRA patterns define several variations in the assignment of responsibilities. As the result of the use case analysis step, the needed collaboration variation in each CRA pattern is selected.

Among CRA patterns, the Manipulate Data(P4) patterns define several interaction variations in each pattern, as shown in Fig. 5. Once a specific pattern is selected from the use case analysis step, proper interaction variation should be selected in several variations. The factor determining a specific interaction variation is the class's relationship to the '?objective' variable in the use case analysis stage has with other classes. The relationship between classes can be grasped through the conceptual class model.

Fig. 7 shows that Updata_Data_1 is selected among the three interaction variations of the Update Data (P4) pattern according to the predefined rule. As shown in Fig. 8, the '?objective' class (Employee) has the updated item '?paramP4'(paymentMethod) as its attribute. So, the relationship between '?objective' class and '?associatedClass' or '?AssociatingClass' is not required to be considered. Thus, the condition highlighted by the red square is satisfied with the given relationship between the 'objective' class and the updated item, '?paramP4'. For this reason, Update_Data_1 is finally selected.

In connecting two collaboration patterns, the most left one among the same lifelines in the two patterns is the connection point, and it is named as “weaving point.” By overlapping the lifeline that becomes the weaving point, the two patterns are connected. While repeatedly weaving the selected patterns, the segmented CRA patterns are composed into a sequence diagram to realize the given flow of events.

Fig. 8 depicts the weaving of two patterns to compose a sequence diagram for the given example flow of events. The patterns selected according to the answers to each question described in Table II are Invocation by a User pattern in P1 pattern and Update Data_1 collaboration pattern among P4 patterns. As shown in Fig. 8, the most left one among the object's lifeline commonly included in the two collaboration patterns is '?objective+?DMType+Form', which becomes a weaving point. The pattern variables marked with the prefix '?' still are denoted in the object names on the top of the diagram or the messages between lifelines as value assignment is not done in this step.

The ‘invoke(param)’message in the Invocation by a User pattern is designed to be substituted by the first message in the firstly connected pattern to the Invocation by a User pattern. Therefore, the ‘invoke(param)’message is substituted by ‘update + ?objective (?paramP4)’ in the weaved sequence diagram in the lower part of Fig. 8.

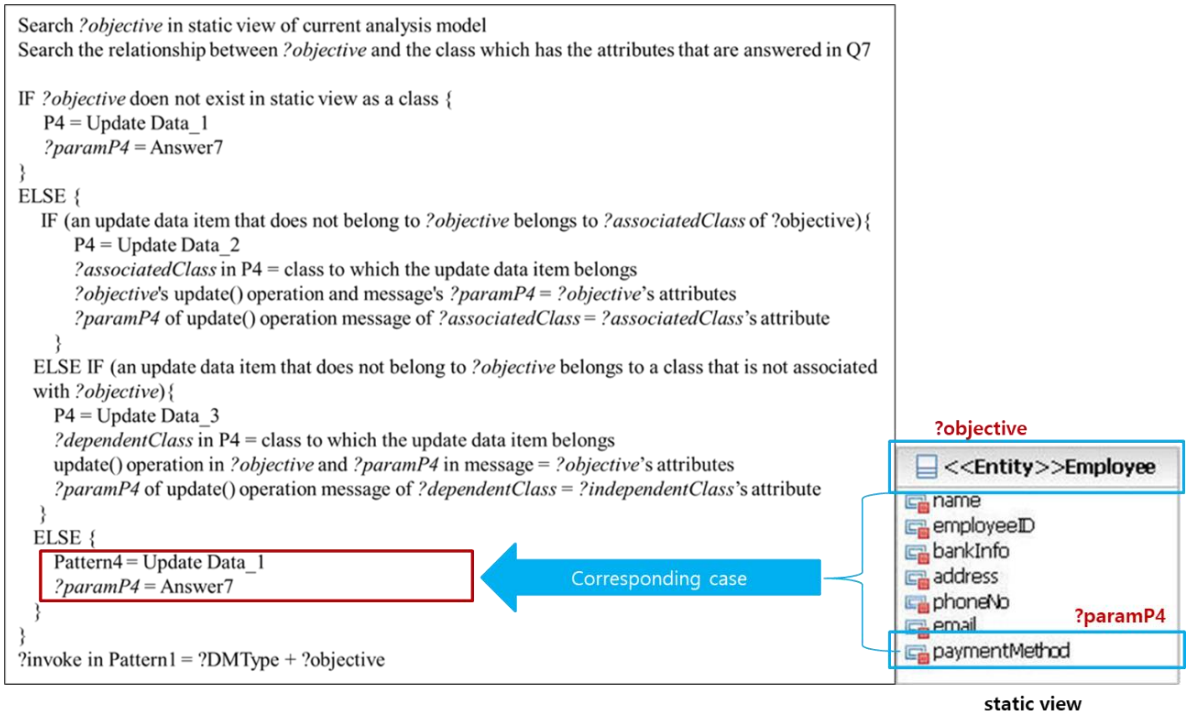


Fig. 7. Rule for Selecting an Interaction Variation in Update Data Pattern.

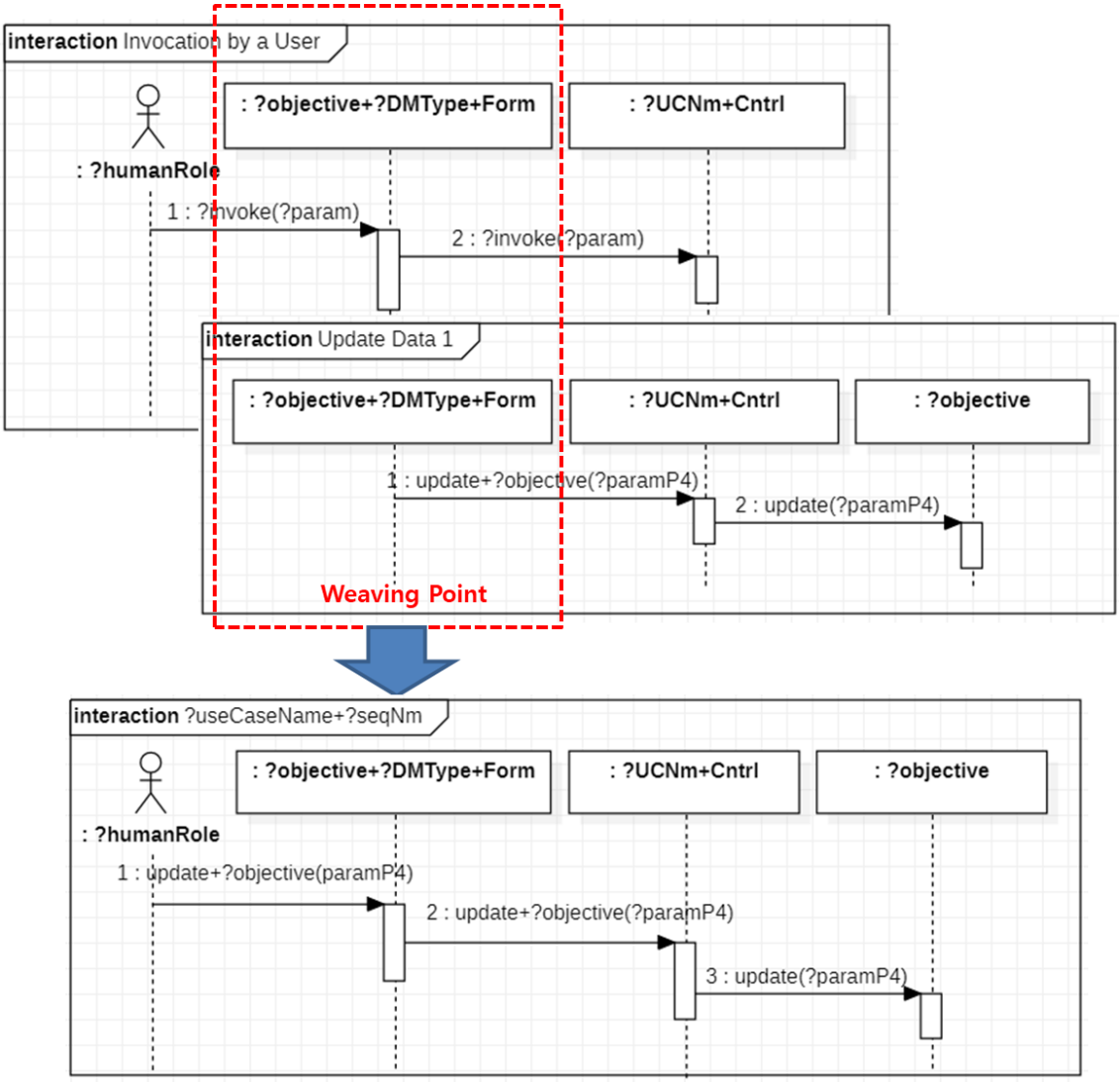
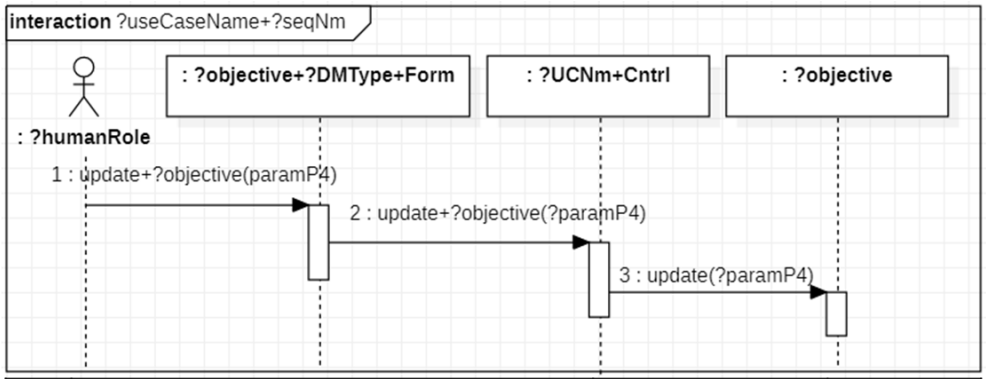


Fig. 8. An Example of CRA Pattern Weaving for Composing a Sequence Diagram for the "Select Payment Method" Flow.

C. CRA Pattern Instantiation

The skeleton of the sequence diagram that realizes the given flow is completed through the CRA pattern weaving step. This step is called CRA pattern instantiation. The upper diagram in Fig. 9 is the weaved sequence diagram for the "Select payment method" flow. Although the sequence of messages is composed, uninstantiated variable patterns exist in the names of an actor, lifelines, and messages. The values to be substituted for the pattern variables included in the skeleton of the sequence diagram are the answers to each question identified in the previous use case analysis step. For example, the name of the control class of this sequence is '?UCNm+Cntrl' is instantiated to 'SlctPmntMthdCntrl' because the extracted value of '?UCNm' is 'SlctPmntMth' according to the values in the table of use case analysis. In the same way, all the pattern variables are instantiated with the values in the table. As a result, the lower diagram in Fig. 9 is completed, with no uninstantiated pattern variable.

The identified responsibility denoted on each message of the sequence diagram should be an operation of the class, which is the message's destination. The developer should keep the consistency between the static view represented by a class diagram and the dynamic view specified by a sequence diagram by adding the identified responsibilities to the proper classes as operations. Fig. 10 shows that the newly identified responsibilities in defining the sequence diagram for the "Select payment method" flow are added to the classes. In building the sequence diagrams with CRA patterns, the newly <<Boundary>>, and the <<Contoller>> stereotyped classes are additionally identified. Comparing the analysis model in Fig. 10 before and after the creation of the sequence diagram, it can be confirmed that the "<<Boundary>>EmployeeUpdtForm" class and the "<<Controller>>SlctPmntMthdCntrl" extracted by the "Update Data" pattern are added to the analysis model.



Q1	What is the Use Case Name?	"Select Payment Method"	?useCaseName
		"SlctPmntMthd"	?UCNm
Q2	What is the name of the flow of event?	"BasicFlow"	?seqNm
Q3	What is the Job characteristic?	"Interactive Job"	invocationType
Q3.1	If it is an interactive job, what is the name of active actor?	"employee"	?humanRole
Q4	What is the data manipulation type?	"Update Data"(Updt)	?DMType
Q5	What is the objective data of the data manipulation?	"Employee"	?objective
Q6	What are the data items to be changed after this update?	"paymentMethod"	?paramP4

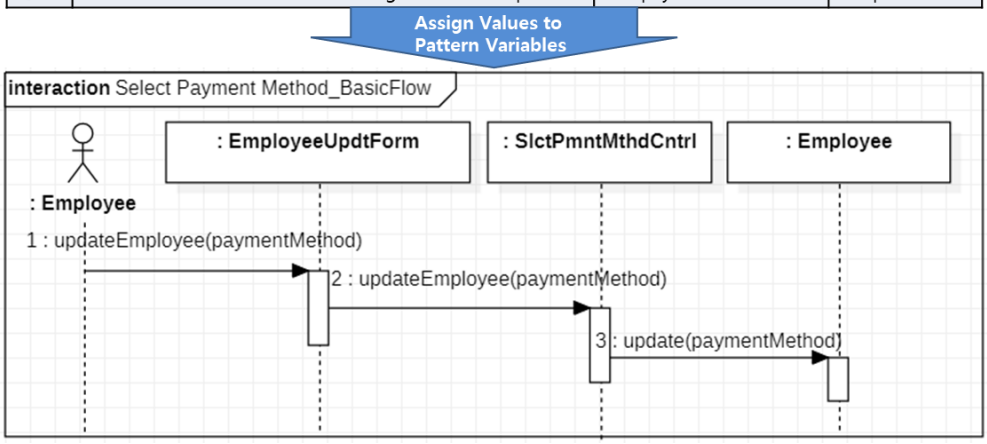


Fig. 9. An Example of CRA Pattern Instantiation to Build the Sequence Diagram for the "Select Payment Method" Flow.

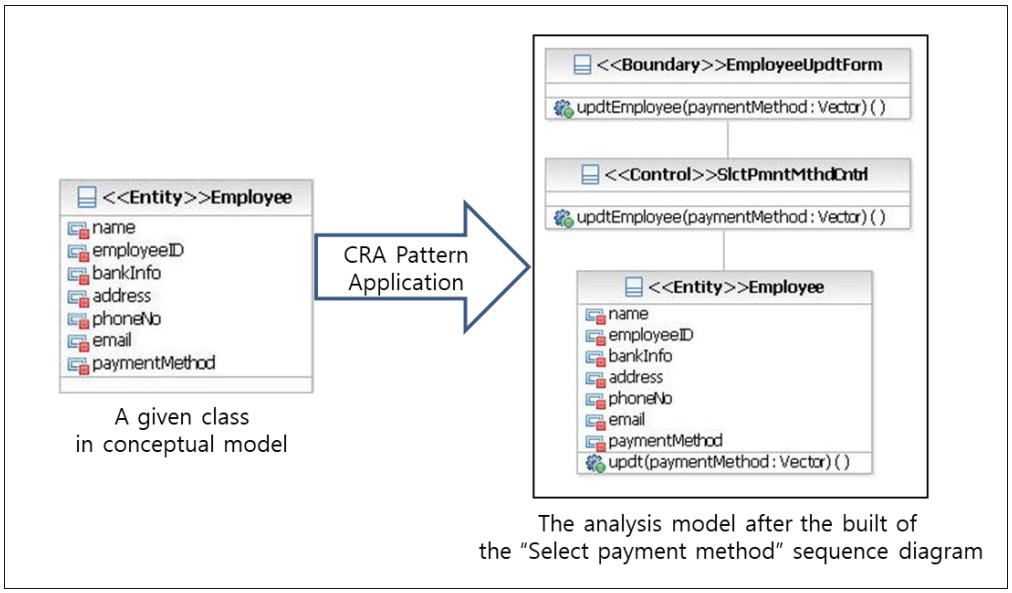


Fig. 10. The Changes of the Static View of the Payroll Management System.

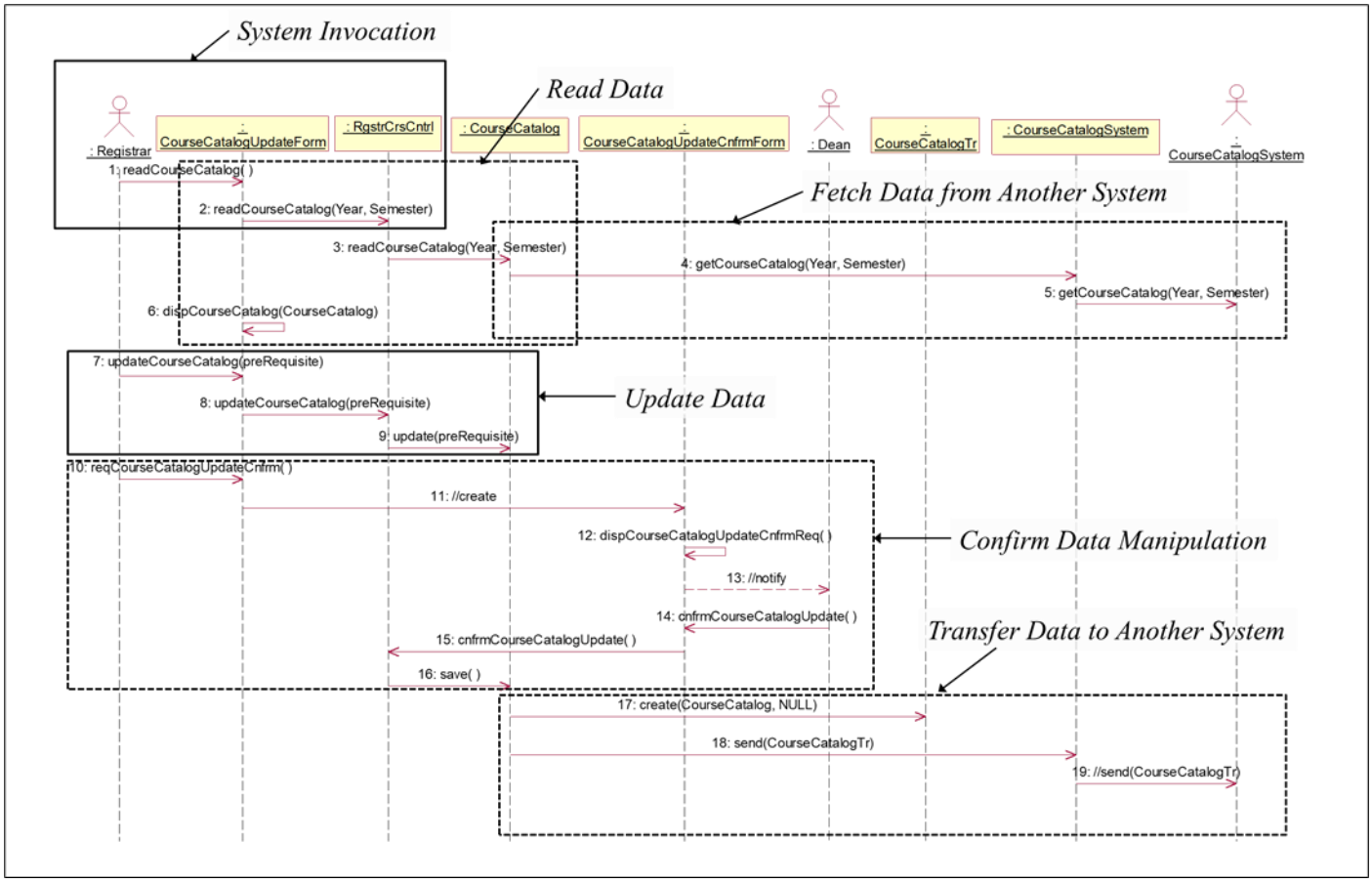


Fig. 11. Applying the six CRA Patterns in Building a Sequence Diagram: the Sequence Diagram for the "Register for Courses" Scenario of the Course Registration System.

D. An Example of Sequence Diagram Built by Applying All the Six CRA Patterns

In proving the feasibility of the proposed CRA patterns, it is necessary to show an example of the sequence diagram to apply all the six CRA patterns. However, unfortunately, in the payroll management system, the target system of this case study, there is no scenario to require all the six segmented CRA patterns. Thus, this study picked one of the scenarios of another system, the "Course Registration System," referenced as an example system in object-oriented analysis textbooks. The selected scenario is the basic flow of the "Register for courses" use case. By weaving the proper set of CRA patterns and instantiating pattern variables with the values from the use case analysis, the sequence diagram in Fig. 11 is built. Similar to the sequence diagram for "Select payment method", Fig. 11 is the sequence diagram with the two required CRA patterns: System Invocation pattern and Update Data pattern. However, in Fig. 11, all of the supporting CRA patterns are also participating. The responsibilities denoted on all messages have been identified from the application of the selected six CRA patterns. This example confirms that the flow composed of considerably long interactions can be realized by applying the proposed CRA pattern language.

VI. Evaluation

Although the case study result shows the feasibility of the proposed CRA pattern language, it is needed to prove how much responsibilities could be extracted from system behaviors in use case specifications by utilizing it. First, this study applied the CRA pattern language to other scenarios in the use case specification of the payroll management system, besides the scenario presented as the case study in Section 5. Those scenarios realized by utilizing the CRA pattern language are: Select Payment Method / Maintain Timecard / Create Employee Report / Maintain Purchase Order / Create Administrative Report/ Maintain Employee Info / Run Payroll / Login.

TABLE IV. Comparison of the Number of Elements in an Analysis Model

Elements # / Type	Conceptual Model	Instantiated Model	Analysis Model
Class	7	35	38
Operation	0	44	51
Attribute	23	27	27
Relationship	6	37	42
Total	36	143	158

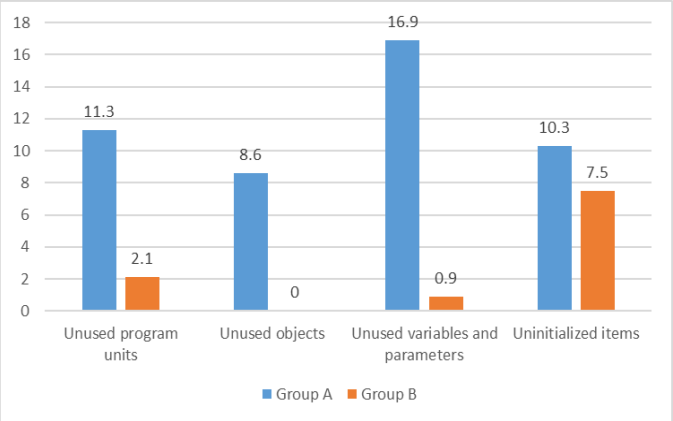


Fig. 12. Comparison of the Bad Symptoms Detected on the Implementation Code.

Table IV compares the number of identified operations (responsibilities) from the CRA pattern application on the scenarios above and the number of all operations in the finalized analysis model to prove the proposed CRA patterns' coverage. The number of design elements extracted purely by the CRA patterns is the value obtained by subtracting the # of elements of the conceptual model from the # of elements of the instantiated model in Table IV. The instantiated model refers to the model obtained as a result of pattern instantiation. The conceptual model is a model that is given as an input, including entity classes identified before applying the CRA patterns. The number of operations included in the conceptual model is 0. After that, the number of operations extracted through pattern application is 44, which is only seven less than 51 operations included in the model at the end of the analysis phase. It means that only seven operations that the developer additionally identified and added to the operation set. Other 86% ((44-0)/51*100) of the class responsibilities of the entire analysis model were identified through CRA pattern application. Likewise, considering the number of whole elements, including classes and relationships, it can be confirmed that 67% ((143-36)/158*100) of the elements are defined as the instantiation of the CRA pattern.

The fact that 67% of the elements of the entire analysis model can be extracted by applying uniform patterns means that, on another axis, 67% of the elements of the analysis model pose the same level of quality. Moreover, applying the proposed CRA patterns implies that GRASP guidelines like high cohesion and low coupling are assured. Thus, it implies that 67% of the analysis model built by the CRA patterns can provide a good and uniform quality even without a separate quality assurance task.

To confirm the effect of applying CRA patterns in the quality of the analysis model, we conducted a controlled experiment. The subjects who participated in the experiment were 4th-year undergraduate students who took the Object-Oriented Analysis and Design course. Students teamed up with 4-5 students to experience from identifying the requirements of the payroll management system to building the application. In constructing the analysis model, only 5 out of 10 teams (group A) provide only use case specifications. To the remaining five teams (group B), a questionnaire for identifying use case specification, CRA pattern specifications, and pattern variables (Table II and Table III) was provided together. That is, in group A, students arbitrarily built an analysis model, and in group B, the CRA pattern language provided in this study was applied to build an analysis model. Both groups completed the development of their payroll management system for ten weeks.

For comparing the quality of the codes written by group B that applied the CRA pattern language to construct an analysis model and group A that did not apply the CRA pattern language, this study conducted static analysis on the implementation code using Understand [22]. As a result, as shown in Fig. 12, it can be confirmed that the number of detected bad symptoms of the source code is significantly smaller in group B than in group A. Among the bad symptom items, the notable result is the number of unused objects/variables and parameters, and those numbers of group B are close to 0. Since the analysis model is constructed by filling the pattern parameters defined in the given CRA patterns with the values extracted from the use case specification, there exists the effect of fundamentally preventing the inclusion of design elements that are not based on the requirements in the analysis model. It is the reason why the number of bad symptoms found in group B is minimal.

The benefits of the proposed CAR pattern language confirmed through the evaluation results can be summarized as follows. The CAR pattern language help that (1) a significant part of the analysis model can be completed by applying the CAR pattern language itself, and (2) developers with little design experience can also be expected to create an analysis model that guarantees consistent quality.

VII. Conclusion and Future Work

The assignment of responsibilities to classes is hard to teach and acquire in practice as many considerations should be taken. Several approaches are proposed to lessen heuristic factors and relieve the efforts to decide which responsibilities are required for a specific class. However, up to now, the existing works, regardless of the used technology, give too general assignment results or too many candidates for one responsibility for developers, which cannot reduce much effort in designing classes.

This study narrows the scope of the proposed CRA pattern language into the business application domain to solve the generality problems. It provides the responsibility assignments results not requiring further revision. The proposed CRA pattern language comprises the six segmented patterns, including several interaction variants according to the relationship format among the conceptual classes.

The six CRA patterns result from vertically decomposing one data transaction performed by a business application into one atomic sequence block. Manipulate Data patterns, which can be seen as the main pattern, contain several interaction variations that specify various collaboration aspects. While searching for the answer to the standardized question set for each flow described in the use case specification, the developers assign values to the variables existing in each pattern. The answers to the questions also determine the set of patterns needed to realize a given flow. The selected pattern

creates a sequence diagram while overlapping the lifeline corresponding to the predetermined weaving point, and this step is called pattern weaving. As a result of pattern weaving, the skeleton for one sequence diagram is completed, and instantiated sequence diagram can be obtained by substituting the values of pattern variables identified in advance.

This study shows the feasibility and the coverage of the proposed CRA pattern language in constructing an analysis model with a case study for constructing an analysis model of a payroll management system. In particular, the results showing that 67% of the operations identified in the final analysis model can be extracted only by applying the proposed CRA pattern proves the differentiation of this study. And, the enhancement of code quality shown through the designed experiment is another benefit of applying the proposed CRA patterns.

The questions for extracting information from use case specifications and the rules for selecting an interaction among the provided interaction variations in a CRA pattern are designed to consider the automation tool development. As for now, the development of the automation tool integrating with a UML authoring tool and Microsoft Word is under construction. With the automated tool, developers just select a proper word to answer a question from use case specifications written in Microsoft Word. And, then, automatically, a proper set of the CRA pattern will be selected, and each word selected by developers will substitute each pattern variable. Consequently, the sequence diagram will be created in a UML authoring tool automatically. Besides constructing the automated tool that supports the CRA patterns, we also plan to extend the case studies to more diverse applications.

REFERENCES

[1] D. Svetinovic, D. M. Berry, and M. Godfrey, "Concept Identification in Object-Oriented Domain Analysis: Why Some Students Just Don't Get It," in *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)*, pp. 189-198, 2005.

[2] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd Edition, Pearson Education India, 2004, pp.736.

[3] M. Bowman, L. C. Briand, and Y. Labiche, "Solving the Class Responsibility Assignment Problem in Object-oriented Analysis with Multi-Objective Genetic Algorithms," IEEE Transactions on Software Engineering, vol. 36, no. 6, pp. 817-837, 2010.

[4] G. Glavas, and K. Fertalj1, "Metaheuristic approach to class responsibility assignment problem," in *Proceedings of the ITI 2011, 33rd International Conference on Information Technology Interfaces*, pp.591-596, 2011.

[5] M. Akiyama, S. Hayashi, T. Kobayashi, and M. Saeki, "Supporting Design Model Refactoring for Improving Class Responsibility Assignment," in *Proceedings of the ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (MODELS 2011)*, pp. 455-469, 2011.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

[7] M. Fowler, Analysis patterns: reusable object models, Addison-Wesley, vol. 10, pp. 357, 1997.

[8] S. Purao, V. C. Storey, and T. Han, "Improving analysis pattern reuse in conceptual design: Augmenting automated processes with supervised learning," Information Systems Research, vol. 14, no. 3, pp.269-290. 2003.

[9] M. E. Fayad, J. Rajagopalan, and A. Ranganath, Stable Analysis Patterns: A True Problem Understanding with UML, 2004.

[10] G. Shu-Hang, L. Yu-Qing, J. Mao-Zhong, G. Jing, and L. Hong-Juan, "A requirement analysis pattern selection method for E-business project situation," in *Proceedings of the IEEE International Conference on eBusiness Engineering ICEBE07*, pp. 347-350, 2007.

[11] X. U. Jin-song, and S. H. I. Lei, "Web application analysis pattern based on recursive MVC structure [J]," Computer Engineering and Design, vol. 12, 2005.

[12] H. S. Hamza, and M. E. Fayad, "The Negotiation Analysis Pattern," in *Proceedings of the EuroPLoP*, 2003.

[13] M. Bowman, L. C. Briand, and Y. Labiche, "Multi-Objective Genetic Algorithms to Support Class Responsibility," in *Proceedings of the 2007 IEEE International Conference on Software Maintenance*, pp. 124-133, 2007.

[14] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Extracting domain models from natural-language requirements: approach and industrial evaluation," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pp. 250-260, 2016.

[15] M. Elbendak, P. Vickers, and N. Rossiter, "Parsed use case descriptions as a basis for object-oriented class model generation," Journal of Systems and Software, vol. 84, no. 7, pp.1209-1223, 2011.

[16] V. B. R. V. Sagar, and S. Abirami, "Conceptual modeling of natural language functional requirements," Journal of Systems and Software, vol. 88, pp. 25-41, 2014.

[17] H. Masoud, and S. Jalili, "A clustering-based model for class responsibility assignment problem in object-oriented analysis," Journal of Systems and Software, vol. 93, pp.110-131, 2014.

[18] M. Albert, J. Cabot, C. Gómez, and V. Pelechano, "Automatic generation of basic behavior schemas from UML class diagrams," Software & Systems Modeling, vol. 9, no. 1, pp. 47-67, 2010.

[19] A. Leff, and J. Rayfleld, "Programming model alternatives for disconnected business applications," Internet Computing, vol. 10, no. 3, pp. 50-57, 2006.

[20] M. Veit, and S. Herrmann, "Model-view-controller and object teams: A perfect match of paradigms," in *Proceedings of the 2nd international conference on Aspect-oriented software development*, pp. 140-149, 2003.

[21] R. T. V. Braga, R. Ré, P. C. Masiero, and C. C. Mourão, "A Process to Create Analysis Pattern Languages for Specific Domains," in *Proceedings of the SugarLoafPLoP*, 2007.

[22] Understand. Available at: https://www.scitools.com/ (accessed 25/08/2021, 2021).