M.Sc. Thesis

A Design of Software Architecture for "SHAPE" Workforce Management Game

An external assignment at IBM Netherlands

By Xiaobo He and Elisabeth E. Mayasari

Software Engineering Group
Department of Computer Science
University of Twente
The Netherlands

Graduation Committee:

Prof. Dr. Ir. Mehmet Aksit (Software Engineering, University of Twente)

Keimpe Zandvliet (IT Organization Consultancy Team, IBM Netherlands)

Ir. Joost Noppen (Software Engineering, University of Twente)

Enschede, August 2003

ABSTRACT

This M.Sc. thesis describes the design of software architecture for the SHAPE (Steering Human Achievement and Purpose Effectuation) Workforce Management Game. SHAPE Workforce Management Game (SHAPE WMG) is a game that simulates the effects of its player's decisions in an environment that can represent the actual business situation of the company where the player works in.

The software architecture of this application is designed with a method called Synthesis-Based Software Architecture Design (Synbad) that is developed by the Software Engineering chair of Computer Science Department at University of Twente. Synbad translates the requirement specification into technical problems, and if necessary decomposes each problem into sub-problems, solve each sub-problem, and integrate the solutions into an overall solution which represents the software architecture. This process involves identifying solution domains for every sub-problem in which the existing knowledge of the relevant domains can be used to form the architecture.

The method starts with requirements analysis that aims to understand the stakeholder requirements and to define the system functional architecture that explains what operations should be performed to meet the system requirements. The process continues with technical problem analysis. In this phase, the requirements are mapped to technical problems. The requirements are generalized and then decomposed into several sub-problems. In the next phase, the solution domain analysis phase, a solution domain model is identified for each sub-problem. Then the knowledge sources are identified for each solution domain. The next phase used in this project is the architecture specification. This phase includes extracting semantics of the architecture and defining the dynamic behavior of the architecture.

After designing the software architecture for the SHAPE WMG, the architecture is evaluated to prove that it is a stable and reusable architecture. The evaluation is done by using the scenario-based architectural analysis where several evolution scenarios are defined for the system.

The project is then continued by implementing the architecture which results in a prototype of the SHAPE WMG. A testing procedure is applied to test and validate the prototype in the final stage of this project.

Through all the steps that are made in this project, the main objective is reached. That is to design architecture for SHAPE WMG that can simulate the effects of its player's decision based on the current condition of the department using Synthesis-Based Software Architecture Design (Synbad) method.

ACKNOWLEDGEMENTS

We are very grateful that we have finally reached the completion of our thesis. We would like to express our gratitude to the following people, without them this thesis would have been impossible and improbable. First, we would like to give our gratitude to the graduation committee: Prof. Dr. Ir. Mehmet Aksit who has given us high academic level guidance on the method and the solutions that are used in this project, Keimpe Zandvliet, Ir. Joost Noppen, and Hans Van Dijk who have given us very useful advice and support, and also kept us focusing on our design. We appreciate the time they have spent to review our design and thesis and to give feedback on our work.

We are also very grateful that we had the opportunity to do our final project at IBM Netherlands, which have brought a perfect ending to our Master of Science study in Telematics. The open atmosphere at IBM and flexible working style has enabled us to learn and work freely. Many thanks to Teunis Westbroek, who helped us in using WebSphere Development Tool and Java techniques during our implementation phase. We also would like to thank Anna van Nouhuys and John Timmermans, who joined the testing session and gave us many useful suggestions.

Xiaobo He would like to thank:

My family and my friends; I always feel lucky that I have such a nice family and a lot of good friends. They always encourage me and offer me great help when I was in trouble or frustrated. Dad and Mom, I love you. Also thanks to Elisabeth Mayasari, Nan Shi, Lin Li, Hong Chen, Rui Wang and Shu Sheng.

Elisabeth Mayasari would like to thank:

My beloved Hananto Setiawan: for his constant care, love, and support. Thanks for believing in me. My dad, mom, brothers and sisters in Indonesia: for the encouragement that has kept me going. Xiaobo "Kevin" He: for such a good work that we have done together. Lina Marliani and Erlangga P. Dharma: for taking really good care of me during the difficult times. My bible study friends: for the constant prayers and the good times that always put me in a better mood. Last but certainly not least, my God: for showing His love and faithfulness. Blessing, honour and glory only unto His name.

August 15, 2003

Xiaobo He and Elisabeth Mayasari

Enschede, The Netherlands

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Background

Over the years, IBM has helped pioneer information technology (IT). With the changes in the industry, its scope and impact has also widened throughout the years: not only to develop hardware, but also to expand their business to the development of software and services. Its existence for over 100 years in this area has made IBM a very much experienced and knowledgeable company. One of the services given by IBM is the Integrated Technology Services (ITS) whose mission is to assist its clients in achieving their e-business goals and ensure that their e-business infrastructure is scalable, available, manageable and secure. ITS provides the people, the processes and the tools to help its clients deliver their expected business results.

The IT Organization Consultancy Team of ITS at IBM has recently tried to develop a new approach to help their clients in understanding several workforce management issues that often come up in their human resource situation. The idea is to let the managers of the client company to play a game that can simulate the effects of their decisions in an environment that can represent the actual business situation in the real world. This game is called "SHAPE" (Steering Human Achievement and Purpose Effectuation) Workforce Management Game. The design and implementation of this project was organized in a Master of Science project which results in the writing of this thesis.

The architecture of this application is designed in this project with a method called Synthesis-Based Software Architecture Design (Synbad). This method is developed by the Software Engineering chair of the computer science department at the University of Twente. The approach used in this method is to translate the requirement specification into technical problems, and if necessary decompose each problem into sub-problems, solve each sub-problem, and integrate the solutions into an overall solution which represents the software architecture. This process involves identifying solution domains in which the existing knowledge of the relevant domains can be used to form the architecture.

1.2 Problem Statement

Professional decisions can be confusing and making an inappropriate decision can cause serious consequences. Making good decisions is one of the most important factors in successfully achieving a company's goal. One of the challenges that a company has to face is how to optimally make use of its available resources to improve their business. A wrong allocation and false use of its resources

can result in high cost expenses with a minimum positive impact, while a better resource planning could give the company a much better result with a much lower cost using the same domain of available resources.

In fulfilling the mission to help its clients deliver their expected business results, the IT Organization Consultancy Team of ITS in IBM came up with the idea to develop a game called SHAPE Workforce Management Game (SHAPE WMG) that aims to help a company to overcome the resource planning problems by evaluating the manager's decision based on the allocated budget, investment decision, and operational cost. This is done by showing the effects of the manager's decision that might lead to several unexpected results. The goal is to make the manager aware of the actions that he makes based on the current situation. The more SHAPE WMG represents the actual business situation of the department, the more things that can be learned by the manager. SHAPE WMG acts as a tool used in the learning process of the decision-making activities of its player in a form of a game. The game support the learning process by simulating the effects of the game player's decisions and giving feedback on why certain things happen caused by the decisions. SHAPE WMG will be played in a workshop given by IBM IT Consultants to their client. The game will be the core material in the workshop instead of the support tool, where the game players, i.e. the IT department managers, are guided throughout the game by the consultants to be able to reach the purpose of the game. Figure 1.1 depicted the interaction between the game player, SHAPE WMG and IBM Consultant that happen in the workshop.

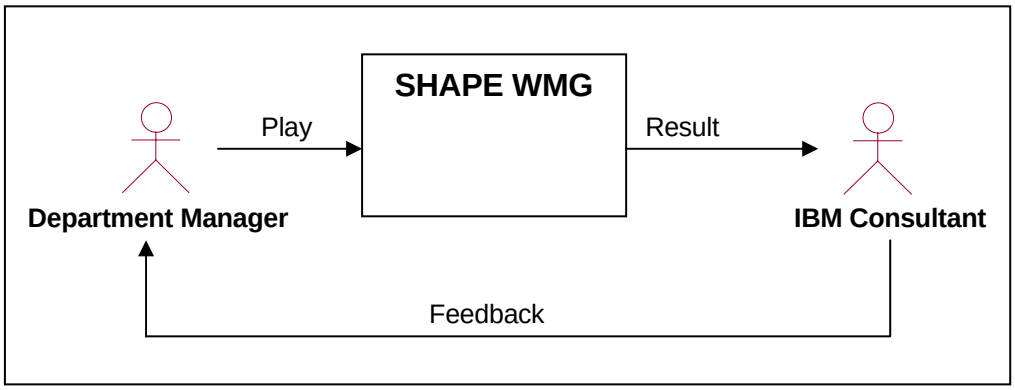

Figure 1.1 SHAPE WMG Workshop

Figure 1.1 shows a department manager that interacts with SHAPE WMG by playing the game. The game receives input from the department manager who acts as the game player. The input represents the player's decisions to achieve the objective of the department based on the current situation of the department that is given in the game. The game then simulates the effects of these decisions. The IBM consultant, who gives the workshop, sees the results together with the game

player and gives feedback to the player on how to make better decisions and to consider some factors that were not yet thought of by the game player.

1.3 The Assignment

The purpose of this MSc project is to design the architecture of SHAPE WMG which is depicted in Figure 1.1. To implement such an application, a software design method is needed. In the start of the project, Synbad was chosen as the software architecture design approach used in developing SHAPE WMG because one of the purposes of the project is to discover the value and applicability of Synbad in an applied research environment. Therefore the application of Synbad method is evaluated at the end of this project to see how well the software architecture that is built using the method can fulfil the requirements. The assessment of the design methodology is to provide assurance of the ease of use of the Synbad method.

SHAPE WMG needs an architecture with which it can create an environment that resembles the real condition of the player's environment, i.e. the department in which the player works. The architecture should also make it possible for the player to make decisions on what to do with the current situation of the department. The effects of these decisions should also be simulated according to the real situations that can happen in the real world. The assessment of the architecture is to see how well the architecture solves all the problems arise in the game playing requirements. But the more important requirement of this project is to create a stable and reusable game architecture that allows future extension and configurations. Therefore, the assessment of the architecture is mainly to prove the stability and reusability aspect of the architecture.

From the description above, we can formulate the main objective of this thesis as follows:
To design architecture for SHAPE Workforce Management Game (SHAPE WMG) that can simulate the effects of the player's decision based on the current condition of the department using Synthesis-Based Software Architecture Design (Synbad) method.

There is a preliminary work of SHAPE WMG which was carried out in the previous year as a four-month project [Jol02]. The work ends in the description of user requirements and a prototype of the game. This thesis continues the work by analyzing the requirements, formulating the problems, finding the solutions and continuing with a design that leads to the implementation of the game.

1.4 The Development Approach

During the project we used an architecture design approach that aims to scope the architecture boundaries from a systematic problem-solving perspective. The method is called Synthesis-Based

Software Architecture Design Approach (Synbad) [Tek00]. There are four main steps in this approach that are carried out in this project:

1. Requirement analysis
2. Technical problem analysis
3. Solution domain analysis
4. Architecture specification

The object-oriented analysis and design methods that we used in applying the Synbad are the Unified Modeling Language [Boo99] and Design Pattern approach [Gam95].

The next step in this project is then to implement the SHAPE WMG based on the developed design. The game is implemented with Java programming language using IBM WebSphere Application Developer ®.

1.5 Outline of the Thesis

The rest of this thesis will be organized as follows:

Chapter 2 describes the background work that is related to this thesis. This chapter gives introduction on Decision Support Systems, several software development methods, and description on what Synthesis-Based Software Architecture Design is and how this design is used in developing software.

Chapter 3 describes the user requirements of the system that is developed, in this case, from the IBM point of view. This will be the basic requirements of the whole process of software development.

Chapter 4 gives the problem and solution domain analysis as part of the steps applied in the design of the system.

Chapter 5 describes the architecture of the system, all the components that are involved in the system and the interaction between them.

Chapter 6 explains the justification of the architecture design described in the previous chapter.

Chapter 7 presents the implementation and testing of the game.

Chapter 8 ends this thesis report by giving conclusions of the project and some suggestions for future work.

Chapter 2

BACKGROUND WORK

2.1 Introduction

This chapter will describe background work that is related to the system that was developed in this thesis. Section 2.2 gives introduction about the definition of Decision Support Systems (DSS) and the three types of DSS. Section 2.3 gives introduction on software development methods and description of several architecture design methods. Section 2.4 closes this chapter with the description of Synthesis-Based Software Architecture Design which is used as the approach in the development of this project.

2.2 Decision Support Systems

Decision Support Systems (DSS) are information systems that support business and organizational decision-making activities. It is intended to help decision makers make use of information from raw data, documents, personal knowledge, or business models to identify and solve problems and make decisions. The structure of the components is shown in Figure 2.1.

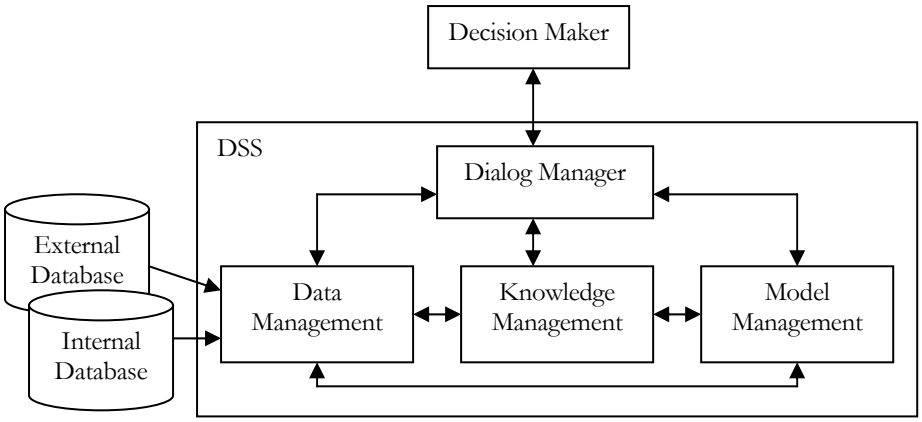

Figure 2.1 DSS Components

There are four main components that build DSS: data management, model management, knowledge management, and dialogue manager. Data management includes internal or external database that contain relevant data for the decision situation. Model management includes software with financial, statistical analysis, graphical, project management, or other quantitative models. Knowledge management provides knowledge for solution of the problem; it either supports the other subsystems or acts as an independent component. The dialogue manager is the user interface that enables the users to communicate with and command the DSS.

DSS exist in many different areas which usually include the following key components: data input, data acquisition system, historical database, and main processing system. In general, there are three types of DSS: Group Decision Support Systems, Executive Information Systems or Executive Support Systems, and Intelligent Support Systems. Sections 2.2.1 to 2.2.3 describe the type of DSS into more detail.

2.2.1 Group Decision Support Systems

Group Decision Support Systems (GDSS) are designed to assist joint decision making process by helping members of the group to share information, exchange ideas and compare alternative solutions. The systems consist of a set of software, hardware, language components, and procedures that support a group of people engaged in a decision-related meeting. The meeting can be in one location or several locations that happens concurrently or at different time. Typical applications of GDSS include email, awareness and notification systems, videoconferencing, chat systems, multi-player games, and meditation systems. The use of GDSS can help improving group productivity and decision quality by improving the members' participation in a meeting. The decision support technologies that are used for GDSS include: decision-modeling methods such as decision trees and risk analysis, structured group methods such as brainstorming, Nominal Group, and Delphi techniques, and rules for directing group discussion. An example of a GDSS software application is IBM's Lotus Notes. With Lotus Notes, the users can have access to a central database, communicate with each other, schedule a meeting, etc.

At the current stage, we do not plan to design a GDSS. At a later stage, however, group decision could be a relevant requirement. For example, SHAPE WMG could be developed to support several department managers to play the game together and make the decisions as group decisions instead of individual decisions.

2.2.2 Executive Information Systems or Executive Support Systems

Executive Information Systems (EIS) are designed to provide information to the top level management of an organization in a highly summarized, convenient, and easy-to-use form. An EIS can facilitate routine management reporting, year-end preview, control and review of major projects, budget preparation, strategic planning, and a general review of the economic outlook. EIS consists of software, hardware, procedure, information, and people. The system gathers, analyze, and integrate internal and external data into information following a set of procedures that is then shown to the executive in a very user-friendly form. It helps the executive find problems and/or opportunities, and then the analysts and middle managers can use a DSS to suggest solutions to the problems and/or what to do with the opportunities. The methods that are used for finding

information needs in EIS include: by-product method, null method, key indicator method, total study method, and Critical Success Factors (CSF) method. Examples of EIS software products are Pilot Software's Command Center and Comshare's Commander Tools.

As EIS, the project we carried out is also designed to help the executives of a company in their decision-making process. Currently, the design does not yet consider providing the executives with access to integrated data that can help them in making decisions. The information given to the department manager is simply numbers that represent several department properties. In the future, it might be desired for SHAPE WMG to provide richer information that can higher the quality in the decision making process.

2.2.3 Intelligent Support Systems

Intelligent Support Systems (ISS) are designed to support decision making with the use of Artificial Intelligence, supported by a combination of databases, knowledge bases, experience and expertise. The systems represent knowledge, offer intelligent advice or take intelligent decisions for the problems found. One example of ISS is Expert Systems (ES). The goal of ES is to imitate human intelligence in solving problems. ES can either support decision makers or completely replacing them. ES combine several human experts from several individual human experts and compile these into broad knowledge bases. Components of an ES are knowledge base, blackboard (the workplace), inference engine (computer program that provides methodology for reasoning), user interface, explanation subsystem (for explaining ES' behavior), and a knowledge-refining system (for analyzing, learning, and improving the system performance). The technologies that are used in ISS include: neural networks, and fuzzy logic.

As mentioned before, ISS is used to replace human experts in making decisions. This can also be the further development that could be applied in SHAPE WMG. The current requirement focus on how the decision making is based mainly on the game player's experiences in handling a certain situation to achieve the objective of the department. In the future, it might be interesting to use Artificial Intelligence in the application to offer intelligent advices that can help the department manager to make decisions and at the same time to learn new strategy in achieving the objective.

2.3 Software Development Methods

A software development method is used to create a software architecture that is able to meet both the functional and non-functional requirements in a precise and constructive manner. In addition, the software architecture should be able to provide good quality of software such as correctness, robustness, adaptability, reusability and maintainability. Today, there are many methods that can be used in developing software. In [Tek00], software development methods are classified as either

artifact-driven, use-case driven, domain-driven or pattern-driven architecture design approaches. Each of these approaches will be generally described in the following sub-sections.

2.3.1 Artifact-driven Architecture Design

An artifact is a general term for any kind of information created, produced, changed, or used by workers in developing a system. *The artifact-driven architecture design approaches are the approaches that extract the architecture description from the artifact descriptions of the method* [Tek00]. A well-known method of this approach is OMT (Object Modeling Technique) [Rum91].

OMT is an object-oriented development technique that consists of analysis phase and design phase. The analysis phase starts with a problem statement that describes the requirement specification. It will then continue with the search and the description of artifacts, the interaction between these artifacts and the methods of the system from the perspective of data flow. The analysis phase generates a set of artifacts instances. The analysis phase is then followed by system design phase, where the overall architecture is defined. The artifacts are grouped into subsystems which form the architectural components. Therefore, the software architecture consists of a composition of subsystems.

2.3.2 Use-Case driven Architecture Design

In a use-case driven architecture driven method, the main focus is on descriptions of typical system usage, known as use cases. In [Boo99] a use case is described as follows: *A use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor.* In this description an actor is an entity that can interact with the system. Such an entity is mostly referred as a role. The use case driven approaches are described in [Tek00] as follows: *The use-case driven architecture design approaches use the use cases as the primary artifacts for deriving the architectural abstractions.* This means that based on the use case model, developers create a series of design and implementation models that realize the use-cases. The developers then review each successive model for conformance to the use-case model.

An example of this approach is the Unified Process [Jac99]. The unified process can be described in two dimensions. According to the first dimension, which is the time dimension, the process is divided into four phases representing the dynamic aspect of the process: inception, elaboration, construction, and transition phases. The goal of the inception phase is to capture all user requirements in the context of use cases. The main activity of the elaboration phase is designing the system in details. At the end of this phase, the design of dynamic and static view of the system is completed. The implementation and the integration of the system are done in the construction phase. The last phase is the transition of the product to its users. The second dimension divides the

process into six core workflows which represent the static aspect of the process: business modeling, requirements, analysis and design, implementation, test and deployment workflows. Use cases play a role in each of these six core workflows that compose the Unified Process.

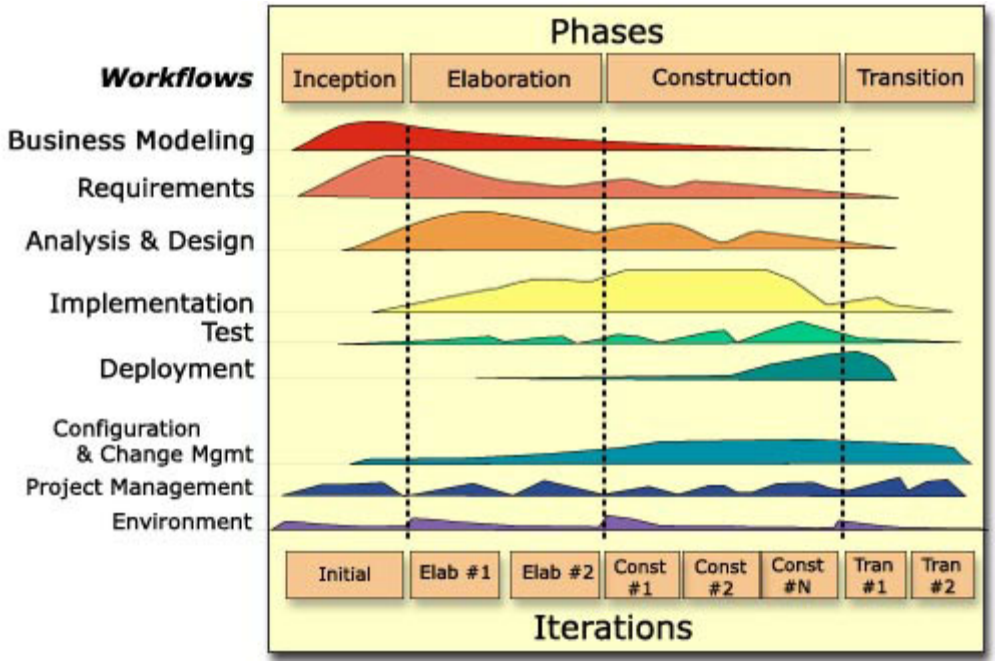

Figure 2.2 Two Dimensions of the Unified Process [Kru01]

In the business modeling workflow, a business model is built to describe the business processes of an organization. The requirement workflow captures the client's requirements as use cases. Use cases are identified to build use-case model which represents the system's behavior. In the analysis and design workflow, the use-cases realizations are created, which describe how the objects interact with each other to help in identifying the artifacts. The identified artifacts are then represented in a design model. During the implementation workflow, the design model is the implementation specification where the use cases are implemented in terms of classes. In the test workflow, the system is verified by performing each use case. The deployment workflow aims at producing product releases and delivering the software to its end users.

2.3.3 Domain- driven Architecture Design

A domain model is a collection of structural information describing the properties of and constraints of a domain. In [Tek00] they are characterized as: *The domain-driven architecture design approaches are the approaches that derive the architectural design abstractions from domain models.* One example of the approach that we will discuss in this chapter is DSSA (Domain-Specific Software Architecture). DSSA consists of domain model, reference requirements, and reference architecture.

The domain model is obtained as the result of the domain analysis phase. The domain analysis phase is done on a set of applications with common problems or functions. The elements of a domain model are customer needs statement, scenarios, domain dictionary, context (block) diagrams ER diagrams, data flow models, state transition models, and object models. Reference requirements are requirements that apply to the entire domain. There are composed of functional requirements, non-functional requirements, design requirements, and implementation requirements. Reference architecture describes all systems in a domain based on the constraints of reference requirements.

2.3.4 Pattern- driven Architecture Design

Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context [Gam95]. *The pattern-driven architecture design approaches are the approaches that derive the architectural abstractions from pattern* [Tek00]. This approach starts with requirement specification that represents a specification of a problem that may be solved using a pattern. Then a search for a suitable pattern is carried on for the given problem description. The search continues with Architectural Pattern Description which describes an architectural pattern. If the rationale for applying a certain pattern is relevant with the given problem and the situation that gives rise to the problem of the pattern matches the situation of the given problem, then the process continues with applying the pattern to the given problem. The result of this phase is Architectural Pattern which will then incorporated to the architecture description. Difficulties might be found with this approach when there is no existing pattern that can solve a particular problem. In this case, a non-pattern solution should be found to solve that particular problem.

2.4 Synthesis-Based Software Architecture Design

In [Tek00] synthesis is described in the following manner: *Synthesis in terms of engineering means a process in which a problem specification is transformed to a solution by first decomposing the problem into loosely coupled sub-problems that are independently solved and integrated into an overall solution.* In other words, synthesis is the process between the concepts Problem Description to Solution Description in the whole software development process. Problem Description is the result of the process of formulating the user requirements into more well-defined problem statement to be able to describe the problem as clear as possible. Solution Description is the representation of the solution suggested to the problems described in the Problem Description. In software engineering the concept Problem Description corresponds to the requirement specification that formulates the client needs in developing certain software applications. The concept Solution Description corresponds to the software architecture design that are ready to be implemented into the real product, i.e. the software. The problem

description is first decomposed into several sub-problems. Every sub-problem is then solved independently before being integrated into the overall solution.

A synthesis-based design process is defined as a finite sequence of synthesis states, resulting in a terminal state [Tek00]. There are two possibilities of terminal state: successful design where solution is found for the problem or unsuccessful design where neither design nor the specification can be modified. This chapter will discuss the synthesis-based software design approach: all of the processes involved in the method and how it is applied in a design of software.

2.4.1 The Process

In Synthesis-Based Software Architecture Design, a distinction can be made between five basic steps, which are depicted in Figure 2.3.

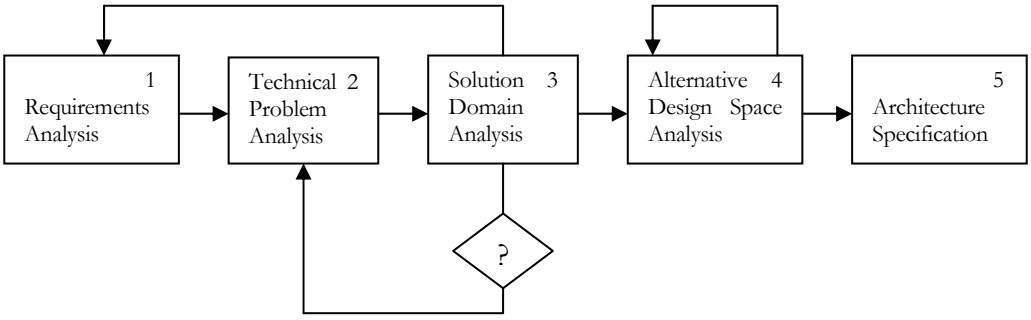

Figure 2.3 Synthesis-Based Software Architecture Design Approach [Tek00]

The lines with arrows in Figure 2.3 connect tasks. The direction of the arrow indicates the sequence of the tasks. The diamond shaped symbol with a question mark represents the validation of a step. The description for each basic process will be explained in section 2.4.2 to section 2.4.6.

2.4.2 Requirements Analysis

The goal of this phase is to understand the stakeholder (e.g. managers, software developers, maintainers, end-users, customers, etc.) requirements and to define the system functional architecture that explains what operations should be performed to meet the system requirements.

This phase usually begins with informal requirement specifications; which can be in the form of textual document as a result of interaction with the client to understand the requirements. From these requirements, the functional requirements of the system are then captured with use cases technique. The use cases are used to model how the system will interact with the users. Scenarios are instances of use cases that represent sequence of actions performed by the system. Afterwards, state transition diagrams can be used to describe the dynamic behavior of the system in terms of services.

2.4.3 Technical Problem Analysis

The next phase is to map the client requirements defined in the previous step to technical problems. First the requirements are abstracted to provide more general and broader view of the problems. Then the generalized problem is decomposed into several sub-problems. Each sub-problem is given a name, initial state and goal. The prioritization and the order of solving the sub-problems is then determined according to the client's requirement or to the solution domain itself, i.e. some sub-problems may required other sub-problems to be solved first before it can be solved.

2.4.4 Solution Domain Analysis

This phase tries to provide a solution domain model that will be used to derive the architectural abstractions. The phase begins with identifying and prioritizing the solution domains for each sub-problem. Every solution domain could have varied range of knowledge sources. Therefore, the next step will be to identify the knowledge sources for each solution domain and to prioritize them based on objectivity and relevancy. After identifying the knowledge sources, the process to gain the knowledge can then begin. The activities involved in this process are to extract the knowledge and then form solution domain concepts to describe the common properties of a set of instances. These solution domain concepts are then structured using *association* relations, where every relation is also derived from the solution domains. Another step in the solution domain analysis is refining the solution domain concepts. This is necessary when a sub-problem has a complex structure that needs to be solved in a more detailed level. The order of sub-problem refinement process is determined by the previously determined prioritization.

2.4.5 Alternative Design Space Analysis

The goal of this phase is to provide a set of possible solutions that can be used for every solution domain concept. First, the alternatives for every concept are defined. If a concept has a complex structure, then it will be necessary to decompose it into several sub-concepts and then define the alternative solutions for each sub-concept. The process continues with describing the constraints between alternatives. This is done to control the number of all possible combination of alternative solutions that could be very large and also to define the right architectural decomposition.

2.4.6 Architecture Specification

This phase begins with extracting semantics of the architecture. This is done for each concept to provide more formal specification. The semantics for an operation of a concept defines the name of the operation, the pre-condition of the concept values prior to the beginning of the operation, and the post-condition of the variables value before the termination of the operation. The next step is defining dynamic behavior of the architecture. Collaboration diagrams are used to illustrate this

dynamic view of the system. It models the interaction between components and therefore shows how the components work together.

Further details and examples on the Synthesis-Based Software Architecture Design approach can be found in [Tek00]. This thesis explains only the general view of the approach and the use of this approach in designing the game application developed during the final project. Due to the demand of implementing the application as soon as possible before the end of the project and the time spent to learn the Synbad approach and several new concepts in developing software, a phase in Synbad is skipped. The phase that is skipped in this project is the alternative design space analysis phase. This means that for every concept formed in the solution domain analysis phase, one solution is offered in the architecture specification phase without seeking for other alternatives solution. This one solution is obtained based on the existing knowledge of the people that are involved in this project.

Chapter 3

REQUIREMENTS ANALYSIS

3.1 Introduction

As mentioned in the previous chapter, the goal of this phase is to understand the stakeholder requirements and to define the system functional architecture that explains what operations should be performed to meet the system requirements. The steps for this phase can be seen in Figure 3.1.

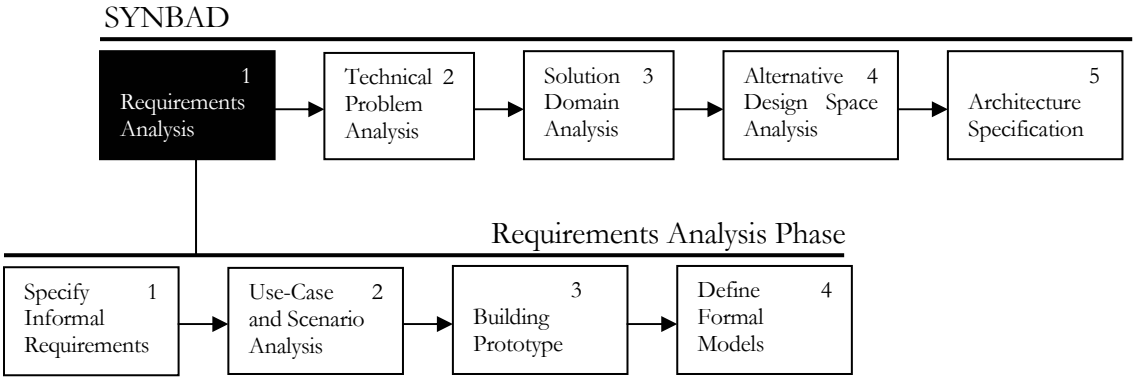

Figure 3.1 Requirements Analysis Phase of Synbad [Tek00]

The phase begins with informal requirement specifications as the starting point in describing the user requirements. These requirements are then described in a more precise and broader perspective by using use cases and scenarios. The step continues with building a prototype based on the user requirements. At the start of this project, there is already a simple prototype that is built using Lotus Approach ®. The prototype gives several ideas on how the user interfaces could be like and the operations that can be done by the user of the application. The last step in this phase is defining formal models. This thesis uses state transition diagrams to describe the dynamic behavior of the system.

This chapter will discuss the requirements analysis phase that are carried on during the SHAPE WMG project. Section 3.2 describes the informal requirements specification, section 3.3 describes the functional requirements of the system using use cases and scenarios, and section 3.4 illustrates the dynamic behavior of the system by using state transition diagram.

3.2 Informal Requirements Specification

The initial requirements of this project were given by IT Organization consulting team of IBM. The idea of this project is to develop a game called SHAPE WMG (Workforce Management Game)

that can simulate the effects of the player's decisions in an environment that can represent the actual business situation of the company where the player works in.

3.2.1 Objective of the Game

The game takes place in a department of a certain company. The department requires several professions. Each profession can be carried out by company's own employees, subcontractors, and/or outsourcers. Every profession in the department has an objective that is represented by a productivity number of that profession. The productivity corresponds to the real netto production that the different type of people produces. The objective of playing the game is to reach the demanded productivity of every profession that exists in the department. This productivity should be reached without exceeding the budget that is allocated to the department. Whether a game player wins the game or not are based on that objective. A game player is said to win the game if he fulfill one of the following condition:

- the game player is able to reach the productivity of most of the professions in the entire game without exceeding the budget
- the game player is able to improve his decision making from the beginning until the end of the game without exceeding the budget at the end of the game and finally reach the productivity of most of the professions

A game player is said to lose the game if he cannot fulfill one of the above conditions.

The game is divided into two main phases: game set-up phase, where the game world is set, and game playing phase, where the game is actually played.

3.2.2 Game Set-Up Phase

For the game to be able to provide current information of the department, then there should be a phase where the game receives those information from a game manager. The game manager is a consultant from the IT Organization consulting team of IBM who obtains this information by interviewing the manager of the client's department. The task of a game manager is to give input on all the necessaries information:

Game-related Information

These are the information that is determined by the consultants themselves in order to give the most suitable learning experience to the managers. The game should be able to allow the game manager to give input to the following information:

- The length of a game period: the beginning of a game period is marked by the player submitting his decisions and the ending of a game period is marked by the simulation of the effects of those decisions.

- The number of periods: represents the maximum number of game periods that can be played from the beginning until the end of the game.
- The disaster for every period: unexpected disturbance that is assigned to every game period. The game manager should be able to choose one of the three possible disturbances: attrition rate change, illness rate change, and budget change and set the changed value.

Business-related Information

These are the information about the current situation of the department that will be played in the game. The information includes:

1. Department information

   The game should be able to allow the game manager to give input to the following department information:

   - For every employee per year: education days, vacation days, learning curve for employees, education cost, recruitment cost, retention cost and golden handshake rate.
   - For the whole department per year: total budget and budget change.

2. Business strategy information

   The game should be able to allow the game manager to give input on the game strategy description that should be applied to the department for every game period.

3. Profession information

   The professions which are included in the game

   - The game should be able to display a list of professions which the game manager can include in the game. We will call this list available professions list.
   - The game should be able to allow game manager to add a new profession that are not in the available professions list or remove a profession from the list.
   - The game should be able to allow game manager to include or exclude professions that are displayed in the available professions list to or from the game.

   For the employees in a profession

   There are three defined types of employee:

   a. Full Time Regular (FTR) employee: an employee of the own company who fulfills a forty (40) men-hours a week.
   b. Full Time Subcontracted (FTS) employee: a subcontractor who fulfills a forty (40) men-hours a week
   c. Full Time Outsourced (FTO) employee: an outsourcer from other company who fulfills a forty (40) men-hours a week

For every profession which is included in the game

There are three defined types of profession:

a. Fixed profession: a profession whose configuration cannot be changed during the game. The number of employees for this profession will remained the same for the whole game. The type of employee in this type of profession is the FTR employee: productive FTR, an employee who gives productivity to the company. The game should be able to allow the game manager to give the following input for this type of profession:
   - For every employee per year: compensation and compensation change per year
   - For the whole department per year: number of employees.

b. Changeable and uncontractable profession: a profession whose configuration can be changed during the game, but whose tasks can only be assigned to the company's own employees. The type of employee in this type of profession is the FTR employee: productive FTR and obsolete FTR, an employee who is idle (doesn't give any productivity). The game should be able to allow the game manager to give the following input for this type of profession:
   - For every employee per year: compensation and compensation change per year
   - For the whole department per year: number of productive and obsolete employees, illness rate, attrition rate, and list of professions whose employees can be reeducated and relocated to this profession.

c. Changeable and contractable profession: a profession whose configuration can be changed during the game and whose tasks can be assigned to subcontractors or outsourcers. The types of employee in this type of profession are the FTR employee (productive FTR and obsolete FTR), the FTS employee, and the FTO employee. The game should be able to allow the game manager to give the following input for this type of profession:
   - For every type of employee (own employee, subcontractor, or outsourcer) per year: compensation and compensation change per year
   - For own employee in the whole department per year: number of productive and obsolete employees, illness rate, attrition rate, and list of professions whose employees can be reeducated and relocated to this profession.
   - For subcontractor and outsourcer in the whole department per year: number of subcontractor and outsourcer.

3.2.3 Game Playing Phase

SHAPE WMG is meant to be played by managers of a company. The decisions that can be made by the game player are related to the configuration of employees in his department. The manager

should use their knowledge and skill to determine how many people to employ or dismiss and whether those employees are from their own company, subcontractors or outsourcers to be able to reach the objective of the company. During the game playing phase, the system should be able to do the following tasks:

- display the length of the game period and the number of periods that are played in the game
- display the business strategy of the current game period at the beginning of the game period to the game player
- display the current department information
- allow the game player to change the education days and/or retention cost values of the department
- display the list of professions that are currently exist in the department
- display the detail information of every profession that are in the list
- allow the game player to hire and/or fire own employee, contract and/or terminate subcontractor and/or outsourcer, and reeducate obsolete employee or from other profession
- allow the game player to change his decisions before he decides to submit all of his decisions as final decisions
- display the result of the player's decisions in the department level
- display the result of the player's decisions in the profession level

Further and more detailed requirements on this project are documented in [Jol02] and [Zan02].

3.3 Use-Case and Scenario Analysis

After studying the basic requirements of the project, the next step is to express these basic requirements in use cases and scenarios to denote the functional requirements. As described in [Boo99], *a use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor*. An actor is the user of the system and therefore is external to the system. A use case diagram models the behavior of the system from the user's point of view. The purpose is to define what the system should do. A use case scenario is an instance of a use case. It describes a particular sequence of activities within a use case. The game is divided into two parts: game set-up and game playing. The use case model and use case scenario will be described for each of these phases.

The use case diagram is depicted using the graphical notations from Unified Modeling Language (UML) [Boo99]. The rectangle represents the system boundary, the stick figures represent actors, and the ovals represent the use cases. The line connecting actor with a use case means that the actors initiate the events involved in that use case. The line with triangle in one end represents

generalization. The triangle is pointing to the superclass. The dashed-line with arrow represents the extend- or include-relationship. An extend relationship from use case A to use case B indicates that an instance of use case B may include the behavior specified by use case A. An include relationship from use case A to use case B indicates that an instance of the use case A will also include the behavior as specified by use case B.

3.3.1 Use Case Model and Scenario for Game Set-Up Phase

The use case model for the game setup phase of SHAPE WMG is depicted in Figure 3.2.

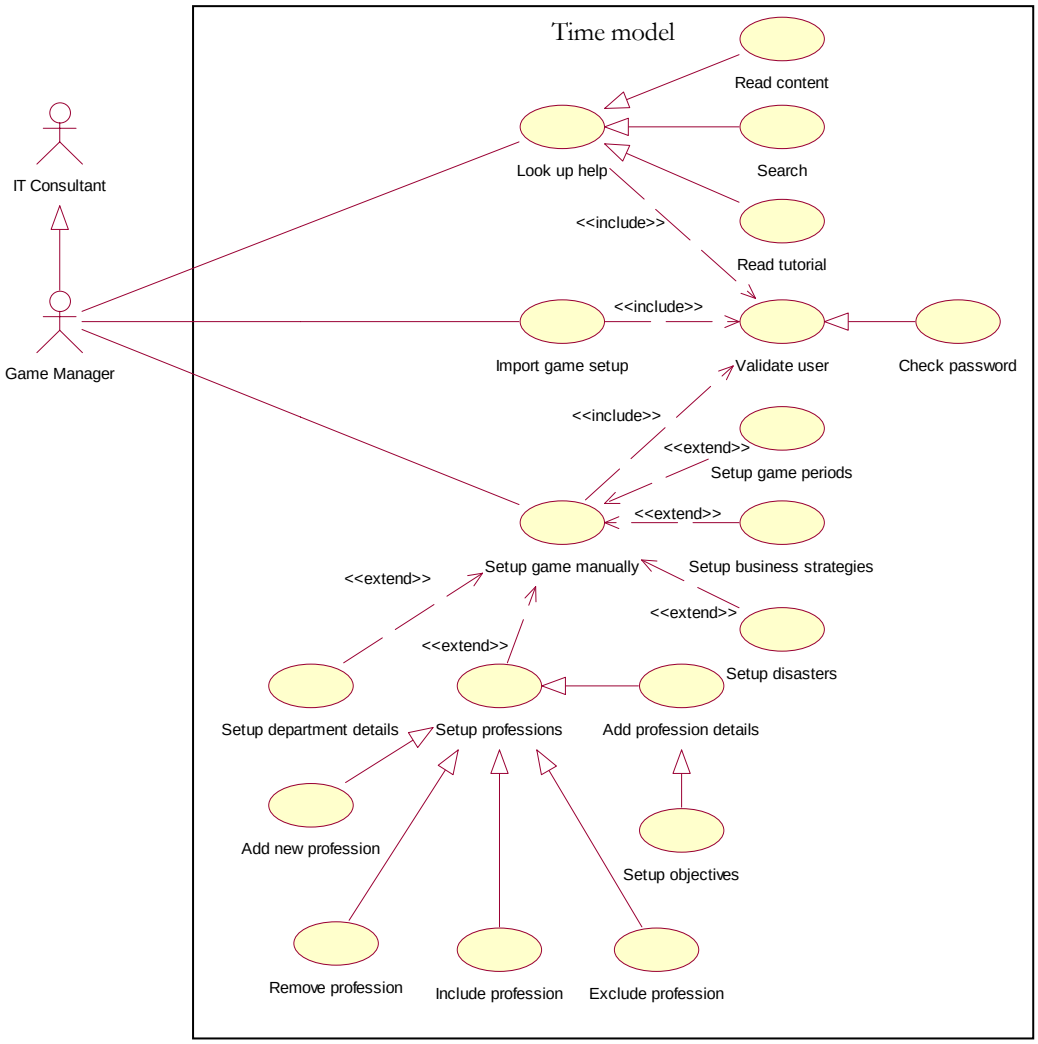

Figure 3.2 Use Case Model for Game Set-Up Phase in SHAPE WMG

Figure 3.2 shows an actor named *Game Manager* that is a specialization of another actor named *IT Consultant*. The *Game Manager* is associated with three use cases: *Look Up Help*, *Import Game Setup* and *Setup Game Manually*. The three of these use cases include use case *Validate User*. This use case is responsible for verifying the identity of the user. The specialized use case of the use case *Validate*

User that is used in the system is *Check Password*. Use case *Check Password* verifies the user identity by checking a textual password.

The use case *Look Up Help* describes the look up help actions: the *Game Manager* can either read the help content (use case *Read Content*), search help using keywords (use case *Search*), or read the game setup tutorial (use case *Read Tutorial*).

The use case *Setup Game Manually* describes the operations needed in setting up all the information one by one: the length and number of periods (use case *Setup Game Periods*), game strategy for every period (use case *Setup Game Strategies*), disaster for every period (use case *Setup Disasters*), the properties of the department (use case *Setup Department Details*), and the professions included in the game and the properties of every profession (use case *Setup Profession*). There are several actions in setting up the profession: add new profession to the available professions list (use case *Add New Profession*), remove profession from the available professions list (use case *Remove Profession*), include profession to the game (use case *Include Profession*), exclude profession from the game (use case *Exclude Profession*), sets up profession details (use case *Setup Profession Details*), and sets up objectives for every profession (use case *Setup Objectives*).

The use case *Import Game Setup* invokes the operation to read a setup file containing all the data to be mapped to the required information. The scenario for Use Case *Import Game Setup* is described below.

Scenarios for Import Game Setup Use Case

Scenario 1: System sets up all the values based on the imported game setup file

1. Game manager chooses the "import game setup" action.
2. Game manager selects a file.
3. System checks to see if the imported file is a valid game setup file.
4. The imported file is a valid file, system then loads the game setup file.
5. The setup values are set to the values that are in the game setup file.

Scenario 2: System fails to set the values since the imported file is not a valid game setup file

1. Game manager chooses the "import game setup" action.
2. Game manager selects a file.
3. System checks to see if the imported file is a valid game setup file.
4. The imported file is not a valid file.
5. System informs the game manager that the file cannot be loaded.

Scenario 3: There are no values set since the game manager cancels the import file action

1. Game manager chooses the "import game setup" action.
2. Game manager cancels the action.

3. System rolls the game manager back to the previous state before the game manager chooses to import game setup.

The scenarios for the rest of the use cases in the game setup phase are described in Appendix A.

3.3.2 Use Case Model and Scenario for Game Playing Phase

The use case model for the game playing phase of SHAPE WMG is depicted in Figure 3.3.

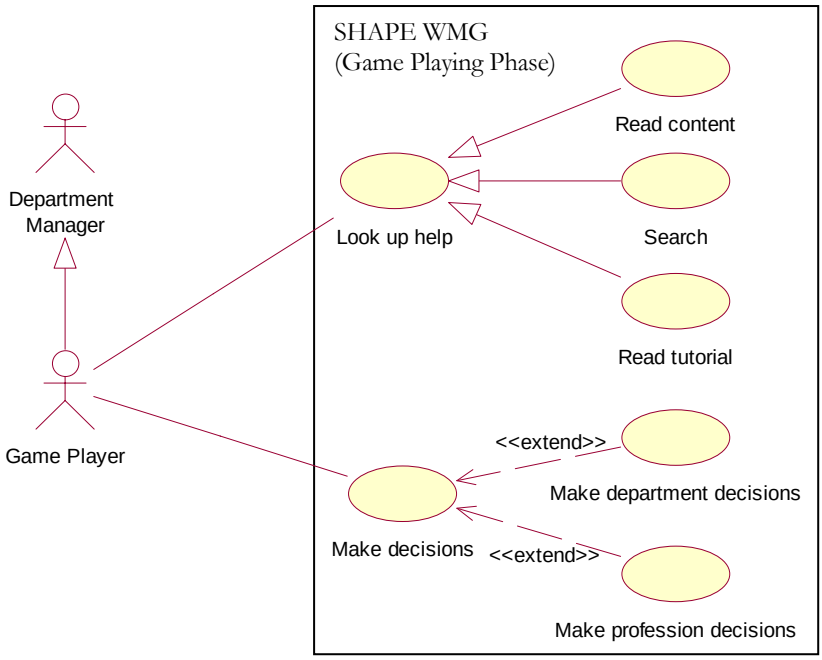

Figure 3.3 Use Case Model for Game Playing Phase in SHAPE WMG

The use case model shows an actor named *Game Player* that is a specialization of a *Department Manager*. The *Game Player* is associated with two use cases: *Look Up Help* and *Make Decisions*. As in the use case model for game setup phase, the use case *Look Up Help* describes the look up help actions: the *Game Player* can either read the help content, search help using keywords or read the game playing tutorial. The use case *Make Decisions* describes the actual game play operations where the game player gives input to the system as their decisions: use case *Make Department Decisions* describes the operation to change several properties of the department and use case *Make Profession Decisions* describes the operation to change the configuration of the professions included in the game. The scenario for every use case is described below.

Scenarios for Look Up Help Use Case

The scenarios for this use case are the same as that of use case *Look Up Help* in the game setup phase. The difference is the content of the help: in the game setup phase the help content is related to the game setup, while in the game playing phase the help content is related to game playing.

Scenarios for Make Decisions Use Case

Normal course:

1. Game player chooses the "start game" action.
2. Game player gives inputs as his decisions.
3. Game player submits the decisions as final decisions.
4. System simulates the results of the game player's decisions.

Alternate course:

1. Game player chooses the "start game" action.
2. Game player gives inputs as his decisions.
3. Game player changes his decisions.
4. Game player submits the decisions as final decisions.
5. System simulates the results of the game player' decisions.

Scenarios for Make Department Decisions Use Case

Scenario 1: System sets new values of the department details

1. Game player chooses the "start game" action.
2. System displays the current details of the department
3. Game player changes education days and/or retention budget of the department.
4. System sets the current values to the values inserted by the game player.

Scenario 2: System fails to set new values of the department details since the game player cancels the action

1. Game player chooses the "start game" action.
2. System displays the current details of the department
3. Game player changes education days and/or retention budget of the department.
4. Game player resets the values, therefore cancels the changes.

Scenarios for Make Profession Decisions Use Case

Scenario 1: System sets profession decisions values based on game player's input

1. Game player chooses the "start game" action.
2. System displays the list of professions that exist in the current game period.
3. Game player selects a profession from the list.
4. Game player choose the "see profession details" action.
5. System displays the current details of the professions including the objective for the current game period.
6. Game player makes decisions by entering the number of employees that are going to be hired or fired or reeducated.

7. System sets the current decisions as temporary decisions.

Scenario 2: System sets profession decision values as zero

1. Game player chooses the "start game" action.
2. System displays the list of professions that exist in the current game period.
3. Game player selects a profession from the list.
4. Game player choose the "see profession details" action.
5. System displays the current details of the professions including the objective for the current game period.
6. Game player makes no decisions to hire or fire or reeducate any employee.
7. System sets the values of the decisions as zero.

3.4 State Transition Diagram

In this phase, we use State Transition Diagram to illustrate the state space of the system and the possible transition from one state to another in the game playing phase. The game playing phase can be divided into three main stages:

1. Make decisions on department level

   When the game starts a new period, it will first display the business strategy of the department for that period. After that, the game player can read the details of the department properties for the current game period. There are two properties of the department that can be changed by the game player: education days and retention cost. The display of the department details also includes the list of professions exist in the department. The game player can choose to read one of the profession listed there.

2. Make decisions on profession level

   After the game player chooses a profession from the list, the game displays the details of the profession properties. Depends on the type of the profession, the game player can then make a decision to reconfigure the current profession. For the configurable type of profession, the following decisions can be made: hire FTR, fire FTR, re-educate obsolete, and re-educate from other profession. For the contractable type of profession, the following additional decisions can be made: make new or terminate FTS or FTO contract. The game player can still change his decisions before he submits them as final decisions.

3. Commit decision and see results

   After the game player submits the final decisions, the system will then show the result of the decisions in the department level. The game player can choose a profession to view more detail results in the profession level. After seeing the results, the game player can then move to the next game period.

This dynamic behavior of the game in the game playing phase is depicted in Figure 3.4. The figure uses the graphical notations from Unified Modeling Language (UML) [Boo99]. Each rectangle with round corners represents a state, which is a point where some events need to take place before an activity can continue. Exceptional are made for the start and the end state. The start state is drawn as a solid black dot, while the end state is drawn as a solid black dot enclosed within a circle. The lines with arrows model the transitions between states. A diamond represents a transition to different branches.

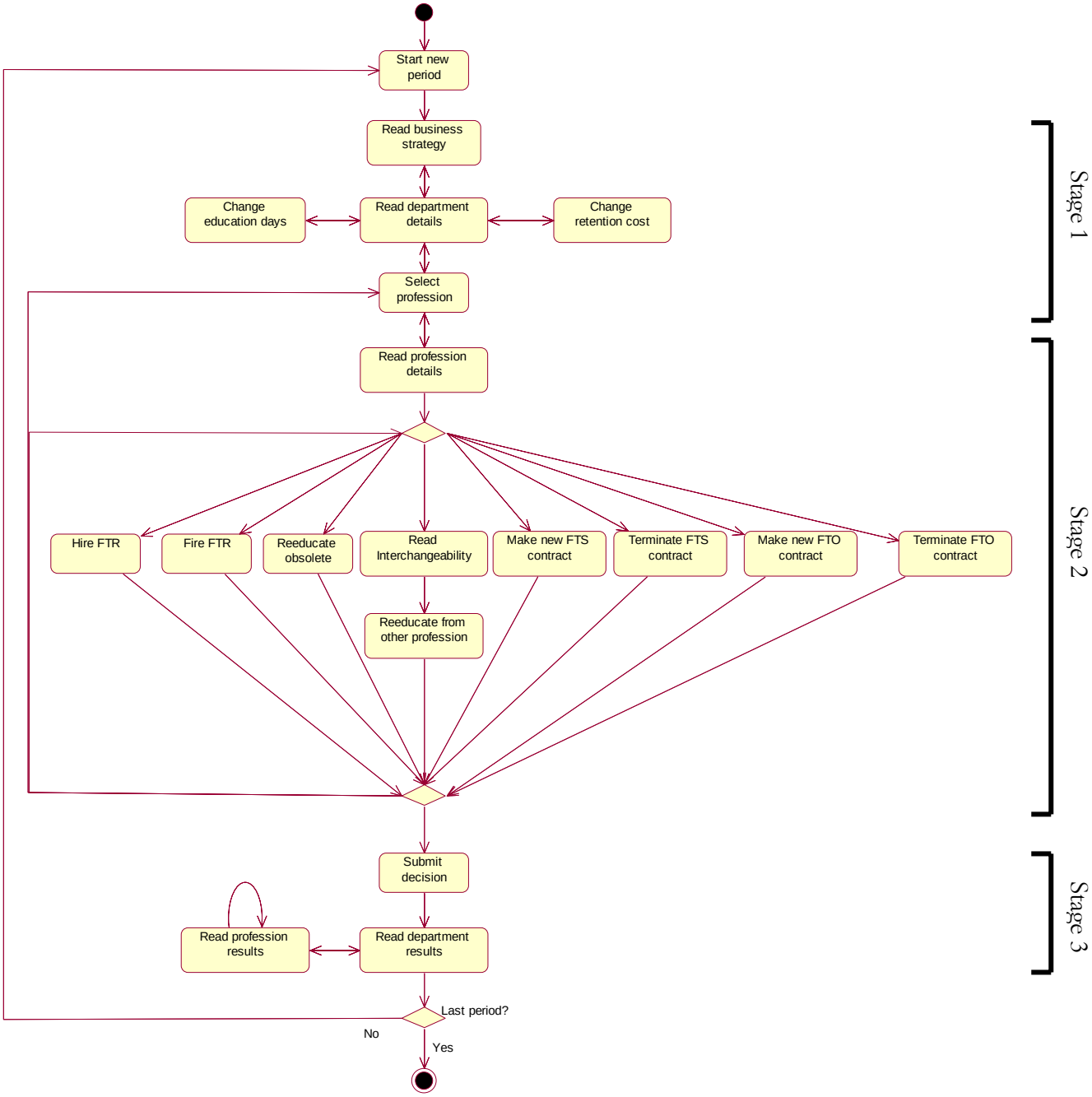

Figure 3.4 State Transition Diagram for SHAPE WMG

Chapter 4

TECHNICAL PROBLEM AND SOLUTION DOMAIN ANALYSIS

4.1 Introduction

After defining the user requirements, the next step in Synbad is to map these requirements to technical problems that describe the actual problems specification to be solved. This phase is called the Technical Problem Analysis. For every sub-problem defined in the Technical Problem Analysis, a solution domain need to be searched. This phase is called Solution Domain Analysis.

This chapter will discuss the two phases and the necessary steps in more detail. Section 4.2 describes the Technical Problem Analysis phase and section 4.3 describes the Solution Domain Analysis phase.

4.2 Technical Problem Analysis

The steps for this phase can be seen in Figure 4.1.

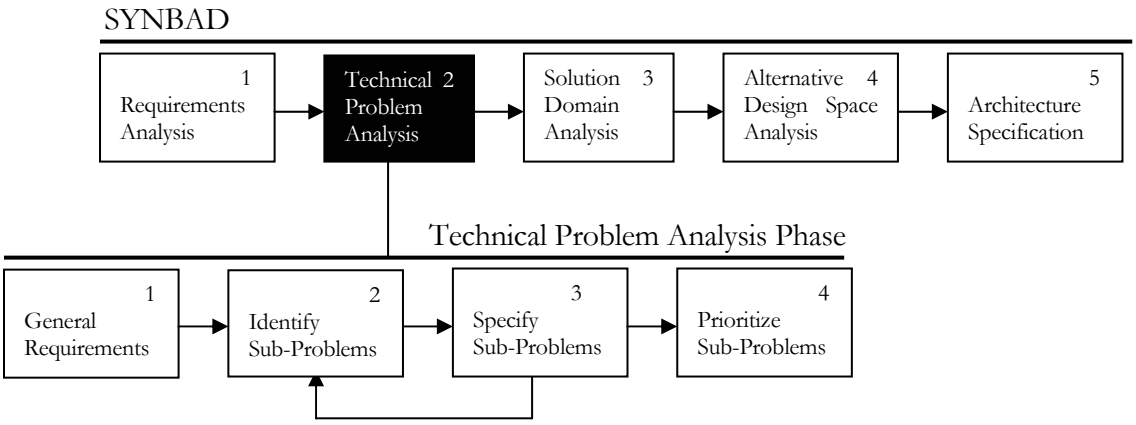

Figure 4.1 Requirements Analysis Phase of Synbad [Tek00]

First of all, the requirement specification is generalized and then mapped to technical problems. If necessary, each sub-problem is then identified and specified. Before moving to solve the problems, however, prioritization is done to determine which sub-problem needs to be solved first.

Subsection 4.2.1 describes the general requirements and subsection 4.2.2 describes the guideline that is used to identify the sub-problems. The identification and the specification of the sub problems are explained in subsection 4.2.3 and the prioritization of the sub-problems is described in subsection 4.2.4.

4.2.1 Requirements Generalization

Referring to the main objective of this project described in section 1.3, the general problem of SHAPE WMG was:

> *How to design architecture for SHAPE Workforce Management Game (SHAPE WMG) that can simulate the effects of the player's decision based on the current condition of the department?*

The technical problems analysis defines every problem with an initial state and a goal that describes the desired state, which is when the problem is solved. The initial state and the goal of the general problem are:

> *Initial State:* There was no application that designed for the purpose of simulating the decisions of a department manager
>
> *Goal:* Design an application that can simulate the effects of a player's decisions in an environment that can represent the actual business situation in the real world

The relevant solution domain for the general problem is simulation game. The solution domain explains that the product of the project is a game that can simulate the input received from its player. In the context of SHAPE WMG, the game represents the department where the game player works in. The game player gives input in the form of decisions that are made to reach the department objective. The game then simulates the effects of the decisions and updates the current condition of the department based on those effects.

After defining the general problem and solution domain for SHAPE WMG, it is found necessary to identify the sub-problems to be able to identify the real problems that will arise in the implementation of the application. The identification of the sub-problems is carried out by using a guideline that is explained in the following subsection.

4.2.2 Guideline to Identify the Sub-Problems

The guideline that is used to identify the sub-problems is by considering the following categorization of problems:

1. Business problems

   The problems in this category concern with the business aspect of developing an application; the key factors to satisfy the client that initiates the development of the application
2. Application specific problems

   The problems in this category concern with the modeling of the fundamental components that are specific to the application being developed.

3. User-application interaction problems

   The problems in this category concern with handling the interaction between the application and the user.

4. Mathematical problems

   The problems in this category concern with designing the mathematical model that is necessary in the application.

5. Computer science problems

   The problems in this category concern with modeling the business solution that refers to knowledge on the computer science solution.

6. Quality requirement problems

   The problems in this category concern with providing application design that is stable and reusable while still is able to provide the proper functionality.

4.2.3 Sub-Problems Identification and Specification

The next step is to identify and specify the sub problems of the general problem described above. Each sub problem is presented with label, name, initial state, and goal. The label consists of the letter 'P' and a number that uniquely identifies the problem. The name describes the name of the problem. The initial state and goal have the same meaning as used in defining the general problem.

Business Problems

- *P1*

  Name: Resources planning

  Initial State: Department managers find problems in making use of the available resources to achieve the department goals.

  Goal: Improve the department managers' skill and knowledge in making decisions for resource planning and make them aware of the possible effects on their decisions.

- *P2*

  Name: Market change

  Initial State: Department managers often not aware of the market condition that change rapidly

  Goal: Increase the department manager's awareness of several type of changes that can happen in the market

- *P3*

  Name: Business strategy

  Initial State: Department managers find difficulties in applying the business strategy of their department to the current situation

Goal: Improve the department manager skill in evaluating the current situation of the department and applying the business strategy based on the current situation

- *P4*

  Name: Periodical decision making

  Initial State: In order to learn decision making process and the effects, department managers would first have to experience the negative effects of their decision in the real world

  Goal: Provide learning experience where the game player has a chance to apply the new knowledge and strategy learned in the previous period to the period after

Application Specific Problems

- *P5*

  Name: Timing

  Initial State: A department evaluates the results of their work in every certain period of time

  Goal: Specify the duration of a period and the number of periods that will be played in the game

- *P6*

  Name: Game world

  Initial State: The game is meant to be played by department managers of the company

  Goal: Provide a game environment that resembles the real condition of the department where the department managers belong to

- *P7*

  Name: Game units

  Initial State: A department consists of several elements that build the department

  Goal: Represent the elements that build the department as the game units that build the game world

- *P8*

  Name: Game rules

  Initial State: In the beginning of the project, there were already game playing rules defined for SHAPE WMG

  Goal: Apply the game playing rules to the related game units

- *P9*

  Name: Game authoring mechanism

  Initial State: There are possibilities to develop the game with new requirements in the future

  Goal: Provide authoring functions to make it possible for the game to be defined incrementally, for example to add or remove game units, activate or deactivate functions

User-Application Interaction Problems

- *P10*

  Name: Interface definition and location

  Initial state: There are two phases needed for SHAPE WMG: game setup and game playing

  Goal: Provide different user interfaces for different needs

- *P11*

  Name: Functions of the interface

  Initial State: Every phase in SHAPE WMG presents different activities

  Goal: Provide different functions for different type of user interface

- *P12*

  Name: Entities modeling of the interface

  Initial State: Every phase in SHAPE WMG can invoke the same or different functions from the same or different game units

  Goal: Define the links between the user interface and the different functions of the game units

- *P13*

  Name: Interface requirements

  Initial State: There was a prototype built to give idea of what the user interface would look like

  Goal: Design user interface that is easy to understand by its user about what is asked from the user to do and provide clear information that are required by the user

Mathematical Problems

- *P14*

  Name: Simulation problems

  Initial State: Several calculations are defined to simulate the effects of the decision

  Goal: Provide realistic simulation that can imitate the real situation in the real world

Computer Science Problems

- *P15*

  Name: System structure

  Initial State: No design structure technique is defined to model the system

  Goal: Model the game by using the system structure that can represent the nature of the game

Quality Requirement Problems

- *P16*

  Name: Evolution problems

  Initial State: The game is expected to evolved in the future

Goal: Provide software architecture design that makes it possible for the system to evolved in the future without having to redesign the whole system

- *P17*

  Name: Performance requirements

  Initial State: The game is expected to function properly according the user requirements

  Goal: Design software architecture that models the requirements of the game

4.2.4 Sub-Problems Prioritization

Every sub-problem is given a priority number that ranges from 1 to 3, with 1 being the highest priority and 3 being the lowest priority. The prioritization of the sub-problems is shown in the following Table 4.1.

Table 4.1 Prioritization of the Sub Problems

ID	Name	Priority
P1	Resources planning	3
P2	Market change	3
P3	Business strategy	3
P4	Periodical decision making	3
P5	Timing	3
P6	Game world	2
P7	Game units	2
P8	Game rules	2
P9	Game authoring mechanism	1
P10	Interface definition and location	2
P11	Functions of the interface	2
P12	Entities modeling of the interface	2
P13	Interface requirements	2
P14	Simulation problems	2
P15	System structure	1
P16	Evolution problems	1
P17	Performance requirements	1

The objective of this project is to design the software architecture for SHAPE WMG. Although the current requirements demand only simple business model to be applied in the game, the design of the architecture is aiming to give stability and reusability that allows future extension and configurations of more complex business model. Therefore, the highest priority in this project is given to the computer science problems, quality requirements problems, and game authoring mechanism problem. The other problems are given lower priorities, which means that in this project the game playing and business aspects of the game are implemented in a very simple way

but the stable and reusable architecture makes it possible to implement the business model and the simulation of the decision making process in a more sophisticated way for future needs.

4.3 Solution Domain Analysis

The steps for this phase can be seen in Figure 4.2.

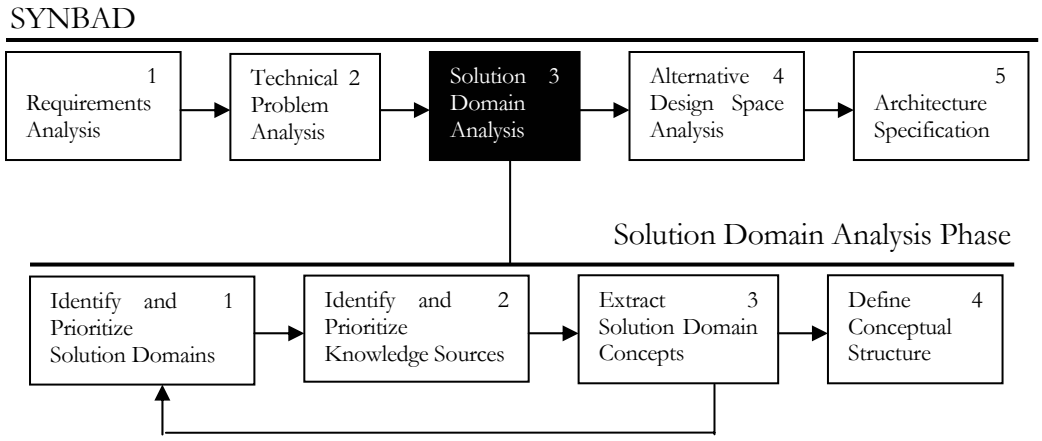

Figure 4.2 Solution Domain Analysis Phase of Synbad [Tek00]

The phase starts with identifying and prioritizing the solution domains for every sub-problem defined in the technical problem analysis phase. Then for each solution domain, knowledge sources are defined and prioritized. After studying and analyzing the solution domain knowledge, the fundamental concepts are extracted from it. The concepts are then structured using relations that are derived from the solution domains. The activities continue with refining the solution domain concepts. This activity is shown in Figure 2.3 as the arrow directed from solution domain analysis phase to requirement analysis phase.

The remainder of this section is organized as follows. Subsection 4.3.1 describes the identification and the prioritization of solution domains and subsection 4.3.2 describes the identification and the prioritization of knowledge resources. The extraction of solution domain concepts, the definition of the conceptual structure, and the refinement of solution domains concepts are not described in this chapter, but is combined with the description of architecture specification which is described in chapter 5.

4.3.1 Solution Domains Identification and Prioritization

For the sub-problems defined in the technical problem analysis phase, solution domains are identified. These solution domains are shown in Table 4.2. As in the prioritization of the sub-problems, the prioritization of the solution domains is defined by a number that ranges from 1 to 3, with 1 being the highest priority and 3 being the lowest priority. The priority was given based on

the considering the importance of solving the sub-problems, which is also related to the prioritization that was made for the sub-problems.

Table 4.2 Prioritization of the Solution Domains

ID	Name	Solution Domain	Priority
P1	Resources planning	Assessment and simulation technique	3
P2	Market change		
P3	Business strategy		
P4	Periodical decision making	Game playing modeling	3
P5	Timing		
P8	Game rules		
P6	Game world	Business modeling	2
P7	Game units		
P9	Game authoring mechanism	Object-oriented design	1
P10	Interface definition and location	User interface	2
P11	Functions of the interface		
P12	Entities modeling of the interface		
P13	Interface requirements		
P14	Simulation problems	Calculation	2
P15	System structure	Control system	1
P16	Evolution problems	Quality management	1
P17	Performance requirements		

The solution domains *Control system, Quality management* and *Object-oriented design* are given the highest priority. The reason is because the design of the application should provide a stable and reusable architecture to support future development, therefore these solution domains should be considered in detail. The solution domains *Business modeling, User interface,* and *Calculation* are given second priorities because it should be built on a reusable and stable architecture. Therefore, these solution domains are considered after defining the stable architecture. The solution domains *Game playing modeling* and *Assessment and simulation technique* are given lowest priority because of the limited time available to do the project and the emphasis of the project to provide stable architecture. Therefore, in this project the implementation is intended to fulfill the minimum requirement of the game. The explanation of each solution domain is described in the following paragraphs.

Solution domain *Control system*

The solution domain *Control system* covers the nature of the game. The game must be able to present the relevant business model to the player, to control the decision-making actions based on the business model by the player, and to simulate the results of the decisions which is also based on the business model. In additional, the stability and robustness of the game architecture are also covered by this solution domain.

Solution domain *Quality management*

The solution domain *Quality management* covers the quality requirements issues. The design of the game should make it possible to develop the game further in the future. This means that the design should be able to keep up with the possible future requirements of IBM in fulfilling their clients' needs. The game should also give the proper performance that helps the IBM clients in their learning process.

Solution domain *Object-oriented design*

The solution domain *Object-oriented design* covers the issues to provide game authoring functions. This means that the main architecture of the game can still be used while continuously modifying, updating, and completing the game components.

Solution domain *User Interface*

The solution domain *User Interface* covers the interaction between the user and the game. Since there are two kinds of user, there should also be two kinds of user interfaces: the game setup interface for the game manager and the game play interface for the game player. The user should be able to understand clearly what is asked for the user to do and how the game works.

Solution domain *Calculation*

The solution domain *Calculation* covers all the calculation that is involved in simulating the effects of the game player's decisions. This calculation should represent the consequences in the real business situations that occur in the real world.

Solution domain *Business modeling*

The solution domain *Business modeling* covers the representation of the business world in the game. For example the representation of a department, professions exist in the department, etc belong to this solution domain.

Solution domain *Game playing modeling*

The solution domain *Game playing modeling* covers the way the game is played. The game should be able to give clear definition on the objective for the game player in playing the game and also on how to reach the objective.

Solution domain *Assessment and simulation technique*

The solution domain *Assessment and simulation technique* covers the issue related to evaluating the behavior of the game player in the decision making process. The game is then able to give feedback to the game player on the steps that were made to reach the objective. The game should also apply simulation technique to give realistic effects that resembles the real situation in the real world.

4.3.2 Knowledge Sources Identification and Prioritization

The next step is to identify and prioritize the knowledge sources for every solution domain identified in the previous step. The solution domain knowledge is prioritized according to the objectivity and relevancy factors. The knowledge source that has high objectivity factor means that it has the detailed and reliable knowledge that can be used to solve a problem. The knowledge source that has high relevancy factor means that it gives the same concern of knowledge that is needed to solve the problem. The knowledge source that has higher objectivity and relevancy factors than the others is utilized first to solve the problem. This knowledge source is then called to have the higher priority than the others.

For the overall solution domain, the knowledge sources are given in Table 4.3. Every knowledge source is presented with the description of its *ID, Knowledge Source,* and *Form*. *ID* gives the identifications of the knowledge source, *Knowledge Source* gives the title of the knowledge source, and *Form* gives the format of the knowledge source. The knowledge sources are ordered according to the priority.

Table 4.3 Knowledge Sources for the Overall Solution Domain

ID	Knowledge Source	Form
KS1	*IBM SHAPE Workforce Management Game External Design [Zan02]*	Document
KS2	*SHAPE Workforce Management Game [Jol02]*	Thesis report
KS3	Meeting with supervisors of UT	Person
KS4	Meeting with supervisors of IBM	Person
KS5	*A QoS-Control Architecture for Object Middleware [Ber00]*	Paper

After studying the knowledge sources for the overall problem, the work continues with identifying the knowledge sources for every sub-problem to look into the overall problem in more detail. However, since searching for knowledge sources is a time-consuming work, there was not enough time to explore all related knowledge sources for the identified solution domains and still be able to finish the project, including the implementation, on time. In order to speed up the work in identifying the knowledge sources, meetings are carried out with the supervisors of the university as the main knowledge source. With the help of their knowledge and experience, solutions for the problems can be reached in time. The knowledge resources for every solution domain can be seen in Appendix B.

Chapter 5

ARCHITECTURE SPECIFICATION

5.1 Introduction

The last step defined in Synbad is architecture specification. The following Figure 5.1 depicts this phase.

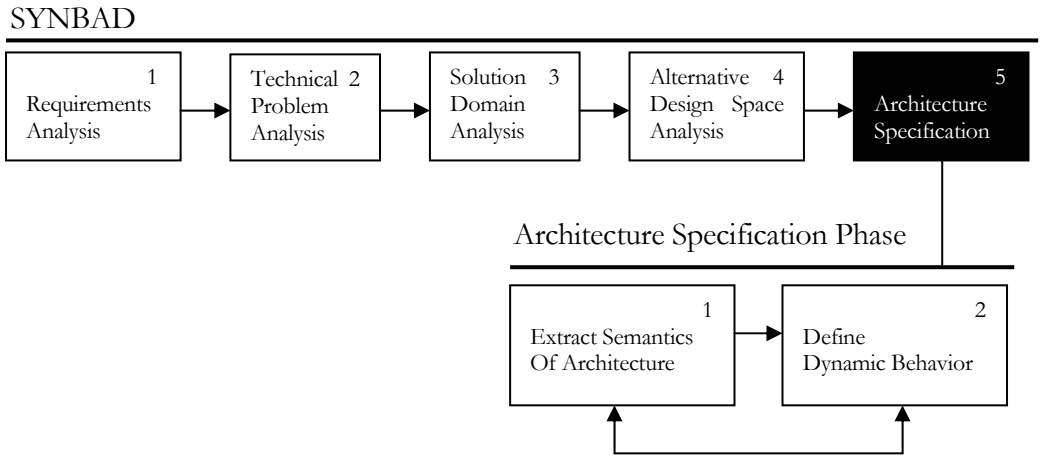

Figure 5.1 Architecture Specification Phase of Synbad [Tek00]

As shown in Figure 5.1, architecture specification phase consists of two sub-processes: extracting semantics of the architecture and defining dynamic behavior of the architecture. The first sub-process derives the semantic of each concept from the solution domains to provide a more formal specification. The second sub-process derives the dynamic behavior of the system from the pre-defined specifications of the architectural components.

Before defining the semantics and dynamic behavior of the architecture, this chapter will first describe the overall and internal architecture of SHAPE WMG by deriving from the solution domains identified in the previous step. From the main objective of this project to provide stable and reusable architecture and also expressed in the prioritization of the solution domain, the dominating architecture of SHAPE WMG. Therefore, this chapter will first describe the generic control system in section 5.2 and then describe the application of the control system in the context of SHAPE WMG in section 5.3. Section 5.4 gives the overall and internal architecture of SHAPE WMG. This chapter will then continue with section 5.5 that focuses on extracting the semantics of the architecture and section 5.6 that describes the dynamic behavior of the architecture. To define the dynamic behavior of the architecture, collaboration diagram are used [Boo99].

5.2 Control System

A control system consists of a controlled system in combination with a controller [Berg00]. The interaction between the controlled system and the controller consists of *observation* and *manipulation* performed by the controller on the controlled system. The building blocks of the control process are shown in Figure 5.2.

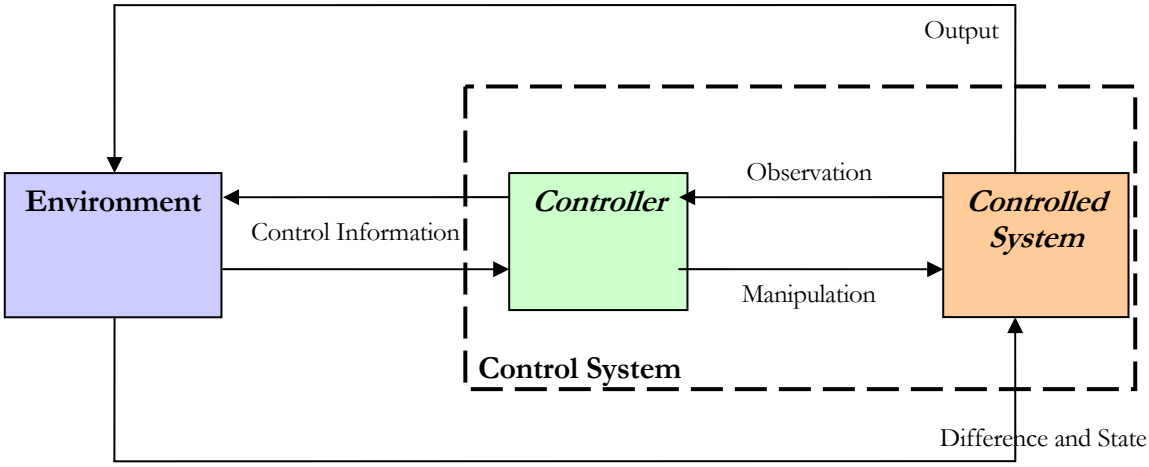

Figure 5.2 Building Blocks of a Control Process

The generic control model abstracts from the type of observation and the type of manipulation that can be employed by the controller on the controlled system. The relationship between the controlled system and the controller can be realised using different strategies. With a *feed-forward control* strategy, manipulation through control actions is determined based on manipulation of the input to the controlled system. A *feed-back control* strategy can be applied for behaviour optimisation. According to this strategy, measurements of the output delivered by the controlled system are compared with a desired behaviour (a *reference*) and the *difference* between them is used by the controller to decide on the control actions to be taken.

5.3 Application of Control System to SHAPE WMG

Referring to the solution domains that were discussed in subsection 4.3.1 and the control system as the dominating architecture for SHAPE WMG, the composition of the solution domains for SHAPE WMG is shown in Figure 5.3.

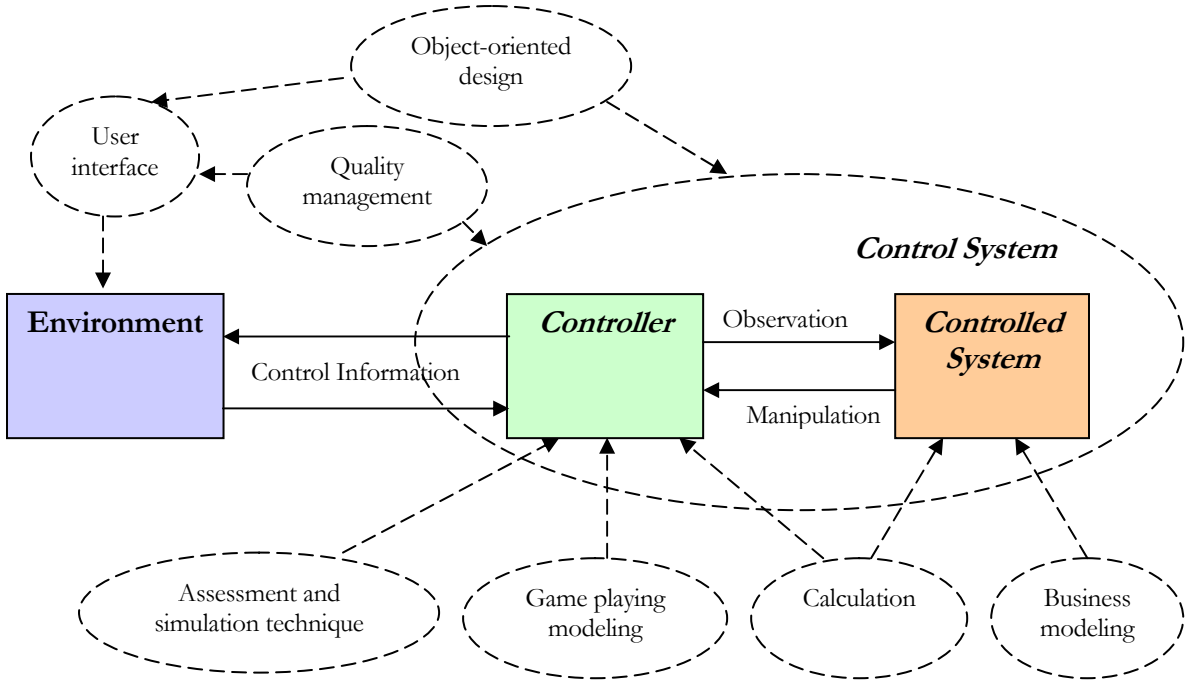

Figure 5.3 SHAPE WMG Solution Domains Composition

Based on the identified solution domains and generic control system theory, the building blocks of the control system in the context of SHAPE WMG are described in the following subsections.

5.3.1 Controller

The Controller of SHAPE WMG is responsible to provide decision making control and optimize the decision making process. The relations between the components are shown in Figure 5.4.

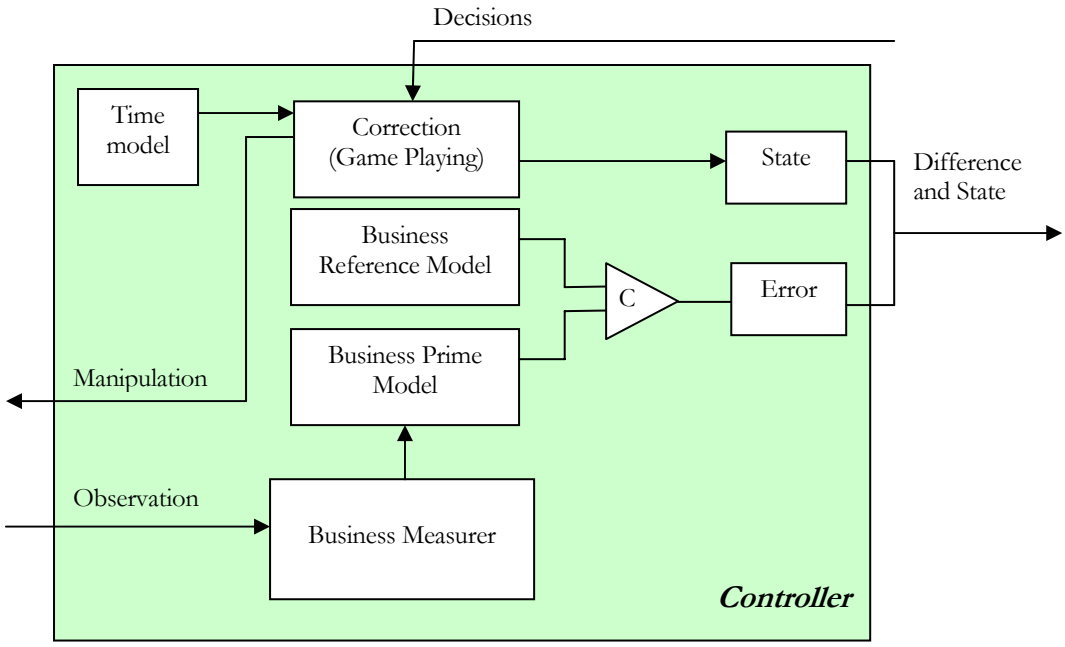

Figure 5.4 Controller of SHAPE WMG

The controller consists of seven components: *Business Measurer, Business Prime Model, Business Reference Model, Comparator, Error, Correction,* and *State*. Business Measurer senses the observation and interprets observation in order to get the measurement. *Business Measurer* senses the observation and interprets observation in order to get the measurement. Afterwards, measurement is sent to *Business Prime Model*, which represents the current business situation. Through C (*Comparator*), comparison has been made between *Business Reference Model* and *Business Prime Model*. The difference is represented by *Error*. Meanwhile, the capturing of the decision making actions are sent to *State*, which describes the state of game playing. Both difference and *State* are sent to *High Level Controller* in order to obtain the advanced control function. New decisions are made by the *Environment* based on the evaluated output from *High Level Controller*. These decisions are sent to *Correction* in order to make an improved manipulation on the Controlled System. In additional, *Time Model* provides time controlling during the game playing. The concept of every component is presented in Table 5.1.

Table 5.1 Concepts of Controller

Sub-Concept	Description of Concept
Business Measurer	The concept *Business Measurer* provides the mechanism for getting observation and interpreting observation into measurement.
Business Prime Model	The measurement of the current business situation makes up Business Prime Model.
Business Reference Model	The predefined ideal business situation is represented by the concept Business Reference Model.
Business Model Comparator	The concept *Comparator* provides the mechanism to compare the same aspects between the business prime model and the business reference model
Business Error	The concept *Business Error* shows the differences found when comparing the ideal business model with the actual business model
Correction	The concept *Correction* represents the game player attempt in correcting the error during the game playing. It could have a collection of control actions.
State	The concept *State* represents the state of a game player in a certain time and the possible decisions that can be made at that point in time
Time Model	The concept *Time Model* models the game that is divided into several game periods with a certain length of time. Which could be a time controlling for the concept Correction.

5.3.2 Controlled System

The Controlled System of SHAPE WMG basically is the *Business Model*, which is used to represent the real business. The components of SHAPE WMG controlled system is shown in Figure 5.5.

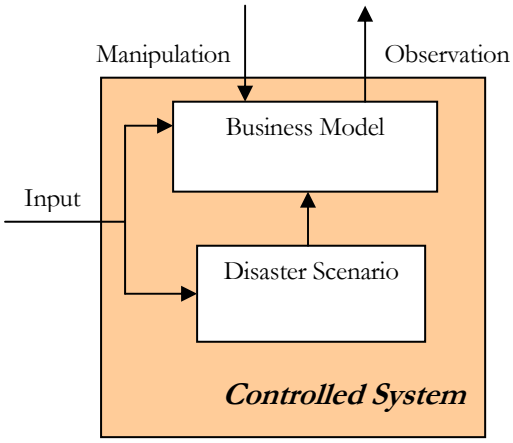

Figure 5.5 Controlled System of SHAPE WMG

Controlled system of SHAPE WMG consists of *Business Model* and *Disaster Scenario*. Input is coming through the Game Manager UI that creates *Business Model* and *Disaster Scenario* during the game setup phase. *Disaster Scenario* will apply disasters to *Business Model*. These disasters are actually set of adjustment values for the department properties in *Business Model*. When *Business Model* and *Disaster Scenario* are up, observation can be sensed by Controller and new manipulation based on the observation will be carried out in *Business Model*. Afterwards, *Business Model* will reorganize itself based the manipulation. The concept of each component of the Controlled System is described in Table 5.2.

Table 5.2 Concepts of Controlled System

Concept	Description of Concept
Business Model	The concept *Business Model* represents the business world in the game whose is created by Environment during the game setup.
Disaster Scenario	The concept *Disaster Scenario* represents the scenario that is created by Environment to add unexpected event (from the player point of view) that can occur during the game.

5.3.3 Environment

Environment consists of two types of user interface. The components of Environment are depicted in Figure 5.6.

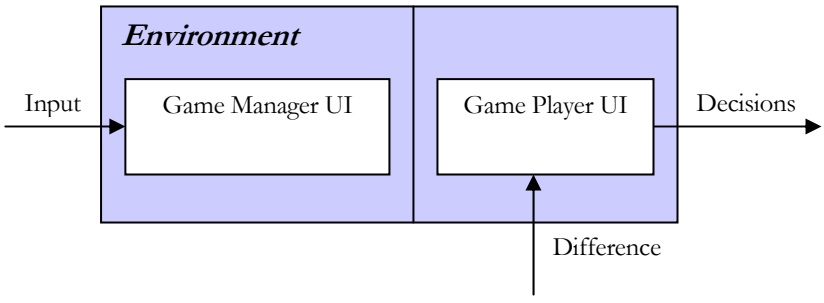

Figure 5.6 Environment of SHAPE WMG

The architecture of Environment of SHAPE WMG models the game users UI, which basically is divided into *Game Manager UI* and *Game Player UI*. The game setup information is sent to Controlled System through *Game Manager UI*, while the decisions during the game playing are sent to Controller through *Game Player UI*. However, *Game Player UI* also gets the evaluated difference from High Level Controller. New control actions (decisions) will be taken based on those evaluated difference. The concept of each component of the Environment is described in Table 5.3.

Table 5.3 Concepts of Environment

Concept	Description of Concept
Game Manager UI	The concept *Game Manger UI* represents the user interface which is between the game manger and Controlled System. Setup input is sent to Controlled System through Game Manager UI.
Game Player UI	The concept *Game Player UI* represents the user interface which is between the game player and Controller. Control information is exchanged through Game Player UI.

5.3.4 High Level Controller

High Level Controller of SHAPE WMG provides advanced decision making control actions. The advanced control actions mainly come from the evaluated difference between Game Player Reference Model and Game Play Prime Model. The components of the High Level Controller are shown in Figure 5.7.

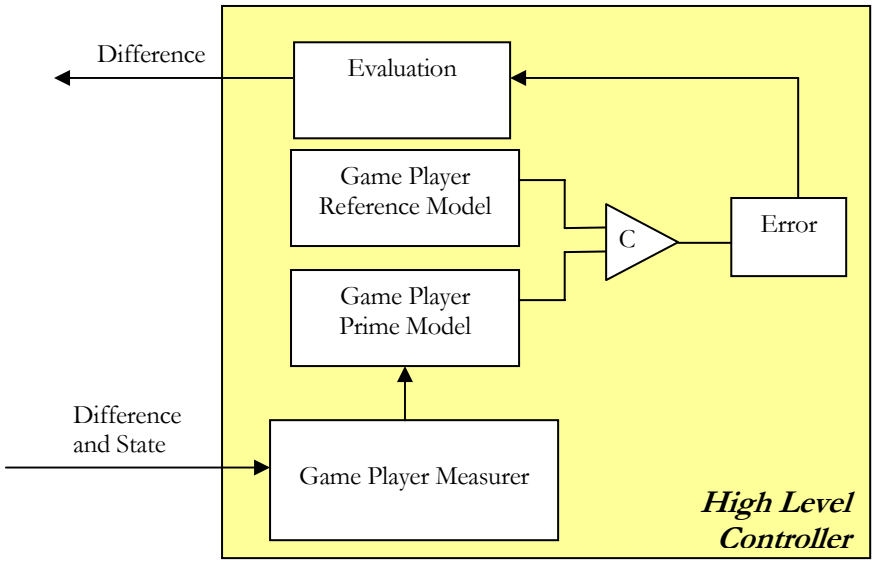

Figure 5.7 High Level Controller of SHAPE WMG

High Level Controller provides an advanced control on game player model that can be modeled by the difference and state, which are sent by the Controller. The difference and state are measured by *Game Player Measurer* and the measurement is used to make up *Game Player Prime Model*. The situation

of current game player is represented by *Game Player Prime Model*. By comparing it with the *Game Player Reference Model*, the difference can be shown as *Error*. After *Error* is evaluated by *Evaluation*, which apply some evaluation rules, the evaluated difference is sent back to Game Player UI and Game Player UI gives input on new improved control actions (decisions) based on the evaluated difference. The concept of each component of the High Level Controller is described in Table 5.4.

Table 5.4 Concepts of High Level Controller

Concept	Description of Concept
Game Player Measurer	The concept *Game Player Measurer* provides the mechanism for getting observation from the lower level controller and interpreting observation into the measurement of current game playing.
Game Player Prime Model	The concept *Game Player Prime Model* models the current game player behavior in making decision in attempt to correct the errors based on the input measurements.
Game Player Reference Model	The concept *Game Player Reference Model* models the ideal game player behavior that could correct the errors. This model is set by the game manager
Game Player Comparator	The concept *Comparator* provides the mechanism to compare the same aspects between the game player prime model and the game player reference model
Game Player Error	The concept *Game Player Error* shows the differences found when comparing the ideal game player model with the actual game player model
Evaluation	The concept *Evaluation* gives evaluation to the game player based on the errors made by the game player through the user interface

5.4 SHAPE WMG Architecture

The overall architecture groups all solution concepts into four parts according to the grouping of the concepts described in the previous subsection. The overall architecture is shown in Figure 5.8. One component that is put outside those four parts is *Help* component. This is the component that provides information to the user about the game. As described before, the game has two phases: game setup phase which involves game manager as the user of the game and game playing phase which involves game player as the user of the game. The architecture will be explained based on the two phases and how the modeling processes are carried on during the phase.

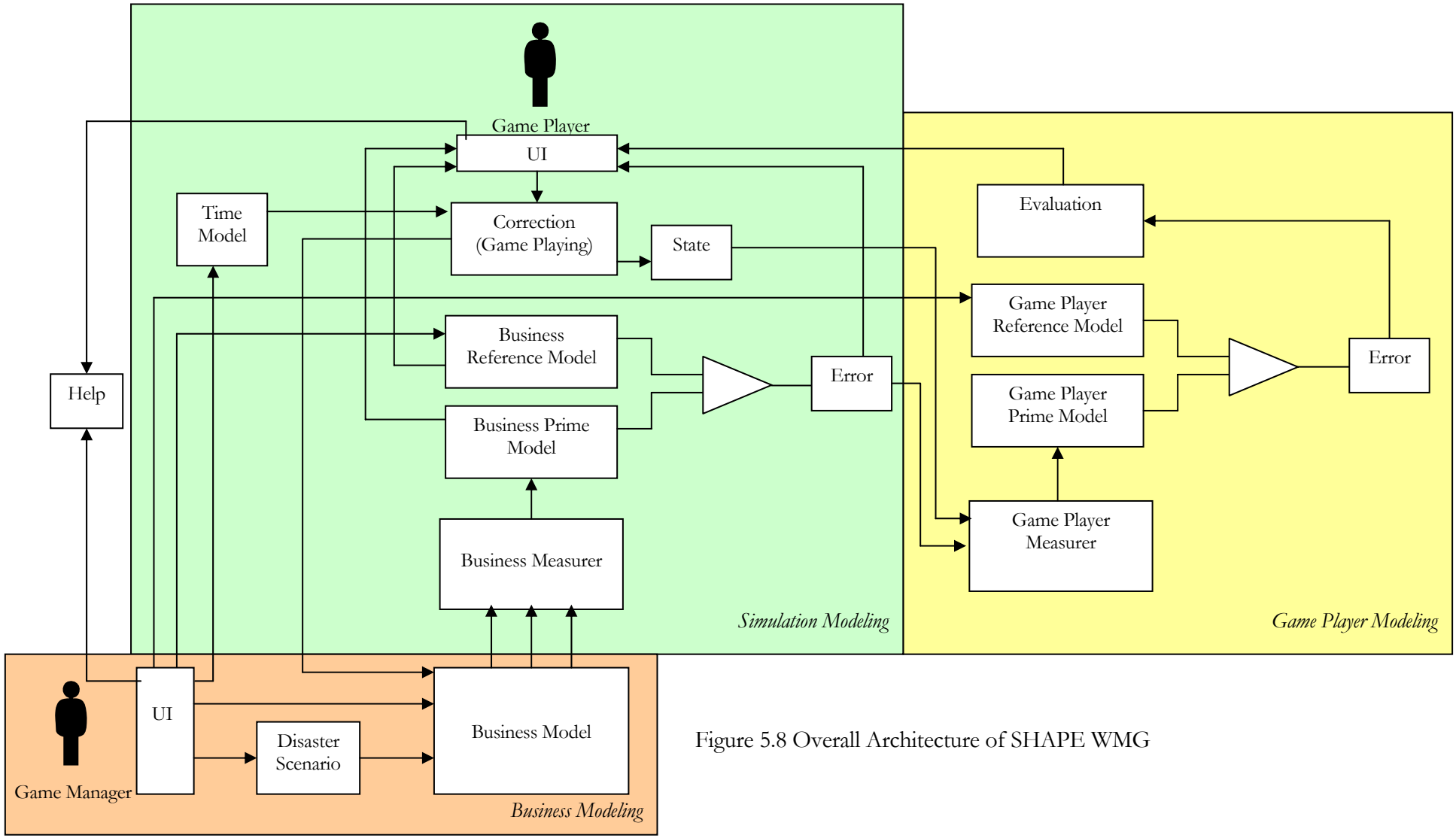

Figure 5.8 Overall Architecture of SHAPE WMG

Game Setup Phase

The following paragraphs will explain how the game setup phase takes part in modeling the architecture parts. Before the game play starts, the game setup should be done first. The game manager sets up the game through the user interface. The input from the game manager could be sent to four different components.

Business Modeling

The primary setup input is sent to the business model, where the business environment, business roles, finances and business rules are stored. The unexpected disasters can be set and sent to the disaster scenario component. The disaster scenarios component gives the surprise aspect to the game player as already mentioned in chapter 3. One disaster can be an adjustment value for an existing setup in a certain game period. The game manager inputs a set of adjustment values for the whole game as the disaster scenarios.

Simulation Modeling

The game manager gives input to the length of the game period and also how many times that period will be played in the game. The game manager also gives input to the business goals and business strategies that are considered to be the ideal business situation. These setups are sent to the business reference model. These values are needed to simulate the result of the game in determining whether or not the player reaches the expected business goals.

Game Player Modeling

The other input of the game manager goes to the ideal game player modeling. This model represents the optimal decisions that can be expected from a game player to make. This ideal game player model is called the reference game player model.

Looking Up Help

During the whole game setup phase, the game manager can look up for help to get information that are related to setting up the game.

Game Playing Phase

After the game setup, the game player starts the game play phase. The game player plays the game through the user interface.

Simulation Modeling

The game is divided into several game periods where each represents a certain period of time. The game player is presented with the current condition of the department (from the business prime model component) and the objective that should be achieved to reach the business goal for the

current game period (from the business reference model component). All game play inputs are sent directly to the correction component as an attempt to make the best decision in order to achieve the ideal business situation. The result of the game player's decisions can only be simulated if the game player submitted his decisions as final decisions. So before the submission is made, the game player can still change his decisions.

Business Modeling

Once the game player submits his final decisions, these decisions are sent to the business model component. The business model applies business rules to its business entities based on the final decisions. The result is then compared to the objective of the current period represented by the business reference model component. The difference between the result and the objective is presented to the game player as a success or failure indicator. The game player is then moved to the next game period and repeats the same process until he reaches the last game period.

Game Player Modeling

Every decision-making action of the game-playing phase in a certain game period is caught by the state component. The game player measurer component measures the errors found in the business model and the decisions made by the game player to correct the errors. This measurement results in a game player prime model that represents the game player behavior in making certain decisions when facing a certain error. This model is compared with a game player reference model that represents the ideal game player behavior that is set by the game manager. The difference of this model is then evaluated. The result of analyzing the different behavior is delivered to the game player through the user interface. The modeling of the game player behavior is out of the scope of this project. However, the implementation of this model can be developed in the future to give the game player more materials in the learning process.

Looking Up Help

The game player can look up for help to get information about how to play the game and how to interpret the result shown in the user interface.

5.4.1 Internal Structure of Business Modeling Component

The internal structure of the *Business Model* combined with *Disaster Scenario* components is shown in Figure 5.9.

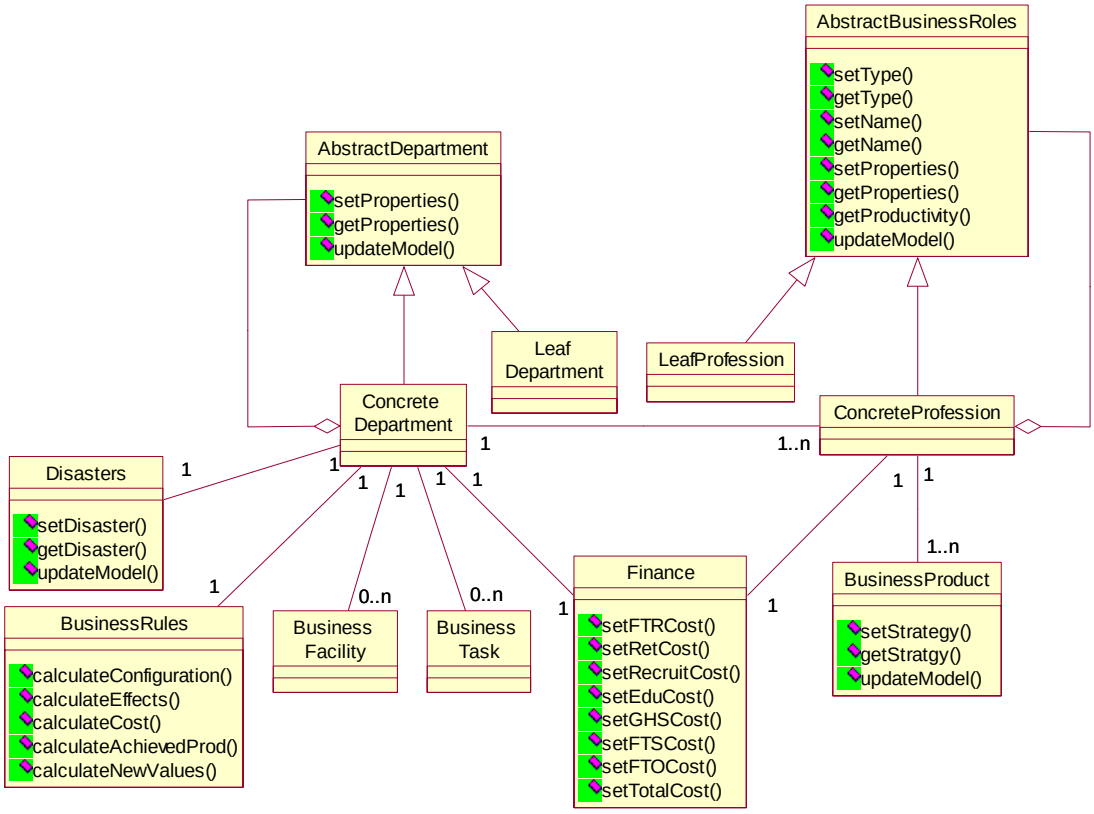

Figure 5.9 Internal Structure of *Business Model* Combined with *Disaster Scenario* Component

Each component of this internal architecture can be seen as a structured concept. There are two main components: *Abstract Department* and *Abstract Business Roles*. The abstract department is the place where the business is running. It generalizes a *Concrete Department*. A concrete department can also generalize one or more other departments. It implements the parent-child relationship. If a concrete department doesn't generalize another concrete department, then this concrete department is called a *Leaf Department*. The same principle applies to the *Abstract Business Role* with the *Leaf Business Roles* and the *Concrete Business Roles*. One department could consist of several business roles. Each business role will produce one or more *Business Product*. Every business role has a *Finance* component that specifies the cost spent for that particular business role for a certain game period. Every department also has a *Finance* component that specifies the cost spent for all business roles that exist within the department. The *Disasters* component, which is used to create exceptional effects on the business, consists of one or more disaster scenarios. A disaster scenario is assigned to a department, one for each period. Since the department is the place where the business is running, *Business Rules* for the running business should be kept in the target department. The specification of each component in this architecture can be found in Table 5.5.

Table 5.5 Specification of Business Model Components

Component	Specification of Component
Abstract Department	An *Abstract Department* defines the basic characters and signatures of a department. It does not have any concrete structures and is never instantiated. From this component, a concrete department can be specified.
Concrete Department	A *Concrete Department* is the specification of a department and inherits the basic characters and signatures of the abstract department. Besides the basic characters are the same, each concrete department can add its own properties and operations. Furthermore it can have sub-departments which are also concrete departments.
Leaf Department	A *Leaf Department* represents a concrete department that doesn't have any sub-department. There is no concrete department that can be specified by this type of department.
Abstract Business Roles	An *Abstract Business Role* defines the basic characters and signatures of a business role In our case it is defined as a profession. The abstract business role has no concrete structure and doesn't represent any real profession. It defines the primary properties and operations. From this component, a concrete business role can be specified.
Concrete Business Role	A *Concrete Business Role* is specified from an abstract business role and inherits basic characters from it. Each concrete business roles have own properties and can specify other concrete business roles.
Leaf Business Role	A *Leaf Business Role* is a concrete business role that doesn't have any specification of other concrete business roles. There is no concrete business role that can be specified by this type of business role.
Finance	A *Finance* component represents the business cost that is spent on the profession and department level. The total budget for the department, the cost of business roles and the cost of each business activities are calculated and saved into this component.
Business Rules	A *Business Rules* component represents the business rules that are applied to the business model based on the game player's decisions done and the properties of the business model. The activities such as firing employee, changing education days can affect the business model. The business properties such as attrition rate, education cost also can affect business model.
Business Product	A *Business Product* is produced by a business role. In our case, the business product is the productivity, which is used to represent the capability of the business role. The productivity is related with some primary factors like: education days, compensation cost of a FTR/FTS/FTO, attrition rate and illness rate, etc
Business Facility	A *Business Facility* represents the facilities and equipments of the department. It could be computers, chairs and development tools etc. In this project, this component is left for further development of the application.

Business Task	A *Business Task* represents the missions that the department has. Tasks are carried out by the business roles. Therefore, business task is indirectly linked to the business roles. In this project, this component is left for further development of the application.
Disasters	A *Disasters* component represents the disaster scenarios that are applied to the business model and set by game manager. Disasters will be used to affect the properties of the department for the target game period. The disasters in our case can be illness rate and attrition rate change.

5.4.2 Internal Structure of Business Reference Model Component

The internal structure of the *Business Reference Model* component is depicted in Figure 5.10.

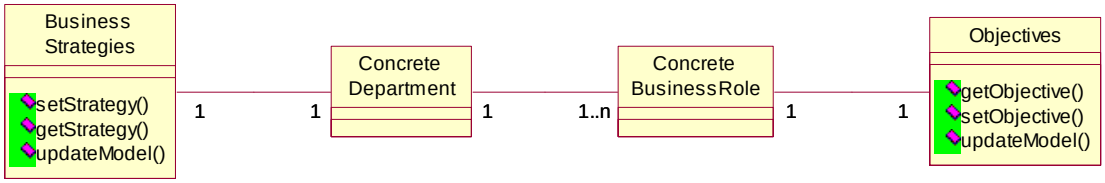

Figure 5.10 Internal Structure of *Business Reference Model* Component

Every concrete department has *Business Strategies* for one or more periods of time. As already been explained before, a concrete department can consist of one or more concrete business roles. Every concrete business role has *Objectives* that represent array of objectives that must be achieved for every game period. The business strategy for a certain period is used to achieve the objective of every profession in that period of time. The specification of each component in the Business Reference Model component is described in Table 5.6.

Table 5.6 Specification of Business Model Components

Component	Specification of Component
Business Strategies	A *Business Strategies* component defines the business strategies of the department for every game period. This strategy is translated into business requirements for the department.
Objectives	An *Objective* component defines the productivities of a certain profession that needs to be accomplished in every game period.

5.4.3 Internal Structure of Correction Component

The internal structure of the *Correction* component is shown in Figure 5.11.

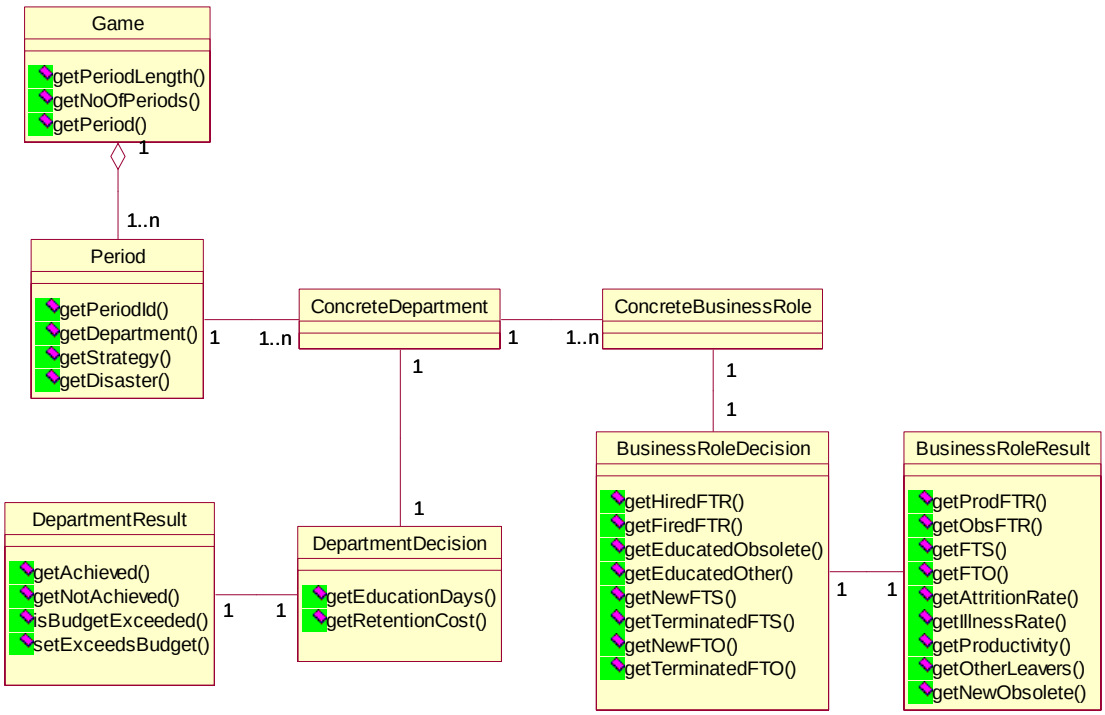

Figure 5.11 Internal Structure of *Correction* Component

The *Game* represents the game that is played by the game player of the SHAPE WMG. The game consists of one or more game *Period*. The number of game period is determined in the game setup phase beforehand. There can be one or more *Department*s that are played in the game. The game player can make decision for every department played which is represented by *Department Decision*. This is the department level decision. *Department Result* represents the result of the game player's decision in the department level. Every department that is played in the game consists of one or more *Business Role*. The game player can also make decision for every profession as the profession level decision, represented by *Business Role Decision*. The result of every profession decision is represented by *Business Role result*. The specification of each component in this architecture can be found in Table 5.7.

Table 5.7 Specification of Correction Components

Component	Specification of Component
Game	A *Game* represents the game that is played by the game player in SHAPE WMG
Period	A *Period* represents a session in SHAPE WMG whose beginning is marked by the game player making decision and the end is marked by the simulation of the decision effects.
Concrete Department	A *Concrete Department* represents the current department situation that is played by the game player.
Department Decision	A *Department Decision* represents the decision that the game player made by changing one or more department properties that are allowed by the game to be changed.
Department Result	A *Department Result* represents the result of the decision that are made by the game player in the department level
Concrete Business Role	A *Concrete Business Role* represents the current profession situation that is played by the game player.
Business Role Decision	A *Business Role Decision* represents the decision that the game player made by reconfiguring the number of employees that are currently exist for a certain business role
Business Role Result	A *Business Role Result* represents the result of the decision that are made by the game player in the profession level

5.4.4 SHAPE WMG Architecture as Decision Support System

To see how SHAPE WMG fits into the structure of Decision Support Systems that were discussed in section 2.2, Figure 5.12 depicts the components of DSS.

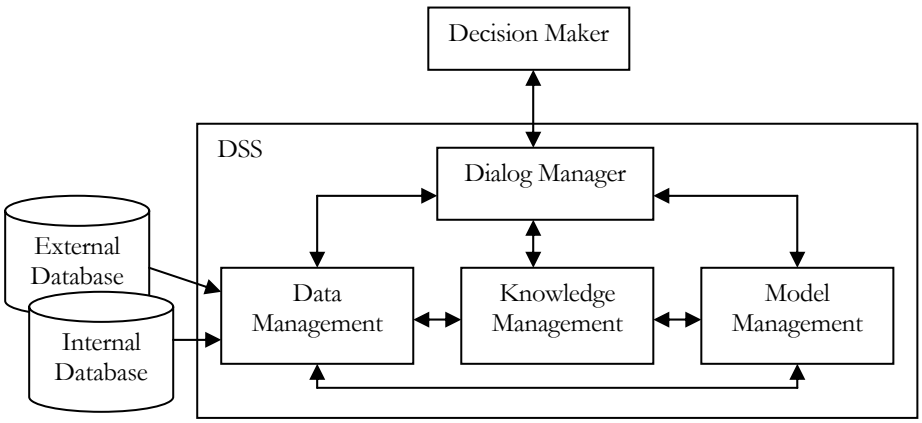

Figure 5.12 DSS Components

The decision maker shown in the figure corresponds to the game player of SHAPE WMG. The dialog manager is the user interface for the game player that connects the game player with the application. The data management corresponds to the business prime model, business reference model than can be accessed by the game player and business model that is updated by the game player decisions. The database corresponds to the object that stores the details of the game units of SHAPE WMG. The knowledge management and the model management are not yet covered by SHAPE WMG, but it is possible to add these components in SHAPE WMG architecture in the future.

5.5 Semantics Extraction of the Architecture

In this phase, the semantics is derived from the solution domain by considering the concepts separately. As mentioned before, this project concentrates on the modeling of the business and the simulation. The modeling of the game player is left out for further development of the application. Section 5.5.1 describes the architecture specification of the game setup phase and Section 5.5.2 describes the architecture specification of the game playing phase.

5.5.1 Game Setup Phase

This section describes the architecture specification of the components and their operations that take part in the Game Setup Phase. Those components are: *Department, BusinessRole, BusinessStrategies, Disasters* and *Objectives.*

Department

The architecture component *Department* represents the department that is played in the game. Example semantics of *Department* is shown in the following figure.

```
Department::setAllProperties(prop: array of integer)
postcondition:
        for (i=0 to noOfProperties - 1)
                properties[i] = prop[i]

Department::setRecruitCost(newCost: integer)
postcondition:
        this.recruitCost = newCost

…
// additional operations
```

Figure 5.13 Specification of the Interface of *Department*

Variable *properties* represents array of the department properties. Operation *setAllProperties* sets all of the department properties to the given values. It is also possible to set only a certain property of the department instead of all of the properties. In the example shown in Figure 5.13 Operation

setRecruitCost sets the value of the *recruitCost*, that represents the recruitment cost for every new employee of the department, to the given value.

BusinessRole

The architecture component *BusinessRole* represents each game period that is played in the game. Example semantics of *BusinessRole* is shown in the following figure.

```
BusinessRole::setAllProperties(prop: array of integer)
postcondition:
        for (i=0 to noOfProperties - 1)
                properties[i] = prop[i]

BusinessRole::setAttritionRate(attrRate:integer)
postcondition:
        this.sttritionRate = attrRate

BusinessRole::setProductivty(prod:double)
postcondition:
        this.productivity = prod

…
// additional operations
```

Figure 5.14 Specification of the Interface of *BusinessRole*

As in the department, operation *setAllProperties* sets all of the business role properties to the given values. It is also possible to set a certain property of the business role. In the example shown in Figure 5.14, operations *setAttritionRate* and *setProductivity* respectively set the value of the *attritionRate* and *productivity* to the given values.

BusinessStrategies

The architecture component *BusinessStrategies* represents the business strategies of the department for every game period. Example semantics of *BusinessStrategies* is shown in the following figure.

```
BusinessStrategies::setStrategies(noOfPeriods: integer, s: array of String)
postcondition:
        for (i=0 to noOfPeriods-1)
                this.strategies[i] = s[i]

...
// additional operations
```

Figure 5.15 Specification of the Interface of *BusinessStrategies*

The *strategies* variable represents array of business strategies for every game period. Operation *setStrategies* sets the strategies of the department to the given description of the strategies.

Disasters

The architecture component *Disasters* represents disaster scenarios that are applied for one more game periods and set by the game manager. Example semantics of *Disasters* is shown in the following figure.

```
Disasters::setDisasters(d1,d2,d3,d4,d5:array of String)
postcondition:
        this.disaster1 = d1, this.disaster2 = d2, this.disaster3 = d3, this.disaster4 = d4, this.disaster5 = d5

…
// additional operations
```

Figure 5.16 Specification of the Interface of *Disasters*

Variables *disaster1, disaster2, disaster3, disaster4,* and *disaster5* respectively represent the disaster scenarios for game period 1, 2, 3, 4, and 5. In the example specification shown in Figure 5.16, there are five disaster scenarios for five game periods. Operation *setDisasters* sets the names and the values of the disaster scenarios to the given names and values.

Objectives

The architecture component *Objectives* represents the objectives of every profession that are set for every game period. Example semantics of *Objectives* is shown in the following figure.

```
Objectives::setObjectives(noOfPeriods: integer, obj: array of integer)
postcondition:
        for (i=0 to noOfPeriods-1)
                this.objectives[i] = obj[i]
…
// additional operations
```

Figure 5.17 Specification of the Interface of *Objectives*

The *objectives* variable represents array of objectives of the business role for every game period. Operation *setObjectives* sets the objectives of the business role to the given values.

Overall Interface Specification

To see how the interfaces described above take place in the architecture design, the components that take part in the Game Setup phase are shown again in Figure 5.18.

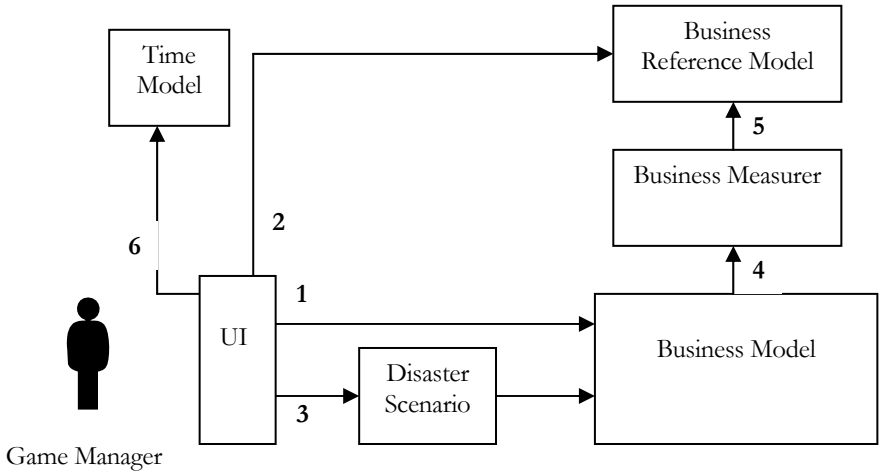

Figure 5.18 Components of the Game Setup Phase

The number shown in Figure 5.18 indicates the interface number to help describing each interface. By referring to the above figure, example of the overall Java interface specification of the game setup phase is described in Table 5.8.

Table 5.8 Interface Specification of the Game Setup Phase

Interface	Java Interface Specification
Interface 1	7. Department.setAllProperties(**int[]** properties) 8. BusinessRole.setAllProperties(**int[]** properties)
Interface 2	4. BusinessStrategies.setBusinessStrategies(**int** noOfPeriods, **String[]** s) 5. Objectives.setObjectives(**int** noOfPeriods, **int[]** s)
Interface 3	1. Disasters.setDisasters(**String[]** d1, **String[]** d2, **String[]** d3, **String[]** d4, **String[]** d5)
Interface 4	1. BusinessRole.countProductivity()
Interface 5	1. BusinessRole.getProductivity()
Interface 6	1. GamePeriodProperties.setPeriodLength(**int** n) 2. GamePeriodProperties.setNoOfPeriods(**int** n)

5.5.2 Game Playing Phase

This section describes the architecture specification of the components and their operations that take part in the Game Playing Phase. Those components are: *Game, Period, DepartmentDecision. BusinessRoleDecision, DepartmentResult, BusinessRoleResult, Finance, BusinessRules, Department,* and *BusinessRole.*

Game

The architecture component *Game* represents the container of the whole game entities that take part in the game playing phase. Example semantics of *Game* is shown in the following figure.

```
Game::createDepartment(dept:Department)
postcondition:
        for (i=0 to noOfPeriods-1)
                Period[i].getDepartment() = dept

Game::createDisaster(dis:Disaster)
postcondition:
        for (i=0 to noOfPeriods-1)
                Period[i].getDisaster() = dis[i]

Game::createDeptDecision(n1,n2:integer)
postcondition:
        Period[periodIndex].getDepartment().getDecision().getEduDays() = n1,
        Period[periodIndex].getDepartment().getDecision().getRetCost() = n2

Game::incrementIndex()
postcondition:
        periodIndex is incremented

Game::start()
postcondition:
        the game is started with first period

…
// additional operations
```

Figure 5.19 Specification of the Interface of *Game*

A game includes an array of *Period* that means that it contains of one or more game periods. The variable *noOfPeriods* represents the total number of periods that will be played in the game. Operation *createDepartment* creates department for each game period all with the same values, which are the initial values of the department in the beginning of the game. A *Disaster* contains an array of disasters for every game period. Operation *createDisaster* sets every disaster to the appropriate game period. The variable *periodIndex* points to the identification of the current game period. Operation *createDeptDecision* creates the department decision component for the current game period and set the values of the department decision on education days and retention cost to the received values. Operation *incrementIndex* increments the *periodIndex* of the game, which means that the *periodIndex* is now pointing at the next game period. Operation *start* starts the game with the first game period.

Period

The architecture component *Period* represents each game period that is played in the game. Example semantics of *Period* is shown in the following figure.

Period::setStrategy(str:String)
postcondition:
this.getStrategy() = str

Period::setDepartment(dept:Department)
postcondition:
this.getDepartment() = dept

Period::setDisaster(disaster:String[])
postcondition:
this.getDisaster() = disaster

Period::addProfession(prof:Profession)
postcondition:
department.getProfessions().get(n) = prof

…
// additional operations

Figure 5.20 Specification of the Interface of *Period*

A period has *strategy* variable that represents the business strategy of the period. Operation *setStrategy* sets the strategy for the period. The variable *department* represents the department that is played in the period. Operation *setDepartment* sets a department to the game period. A period has *disaster* variable that represents the disaster scenario that is assigned to the game period by the game manager during the game setup phase. A disaster scenario contains of a name (attrition rate change or illness rate change) of the disaster and the value of that disaster. Operation *setDisaster* sets the given disaster values to the game period. Operation *addProfession* adds a profession that exists in the department in that game period. In this project, it is assumed that all professions exist from the beginning until the end of the game. In the future, it should be possible to determine the period where a profession starts to exist in the game, e.g. professions that will be needed by the company in the future.

DepartmentDecision

The architecture component *DepartmentDecision* represents the game player's decision in the department level. The semantics of *DepartmentDecision* is shown in the following figure.

DepartmentDecision::getEduDays(): integer
precondition:
decision on new number of education days is defined

DepartmentDecision::getRetCost(): integer
precondition:
decision on new retention cost is defined

Figure 5.21 Specification of the Interface of *DepartmentDecision*

A department decision has two variables: *eduDays*, representing the new number of education days, and *retCost*, representing the new retention cost. If the game player decided to keep the initial values and did not decide to change them, then the values of *eduDays* and *retCost* are set to those initial values. The operations *getEduCost* and *getRetCost* respectively returns the decisions made on the number of education days and on retention cost.

BusinessRoleDecision

The architecture component *BusinessRoleDecision* represents the game player's decision made for a certain profession. Example semantics of *BusinessRoleDecision* is shown in the following figure.

BusinessRoleDecision::setValues(n1,n2,n3,n4,n5,n6,n7:integer)
postcondition:
this.hiredFTR = n1, this.firedFTR = n2, this.eduObs = n3,
this.newFTS = n4, this.terFTS = n5, this.newFTO = n6, this.terFTO = n7

…
// additional operations

Figure 5.22 Specification of the Interface of *DepartmentDecision*

A business role decision has seven variables: *hiredFTR*, representing the number of FTR that are hired, *firedFTR*, representing the number of FTR that are fired, *eduObs*, representing the number of obsolete FTR that are reeducated, *newFTS*, representing the number of new FTS contract, *terFTS*, representing the number of FTS contract that are terminated, *newFTO*, representing the number of new FTO contract, and *terFTO*, representing the number of FTO contract that are terminated. Operation *setValues* sets all of those variables to the given values. If the game player decided not to make decision on any of the variables, those variables will be set to zero.

DepartmentResult

The architecture component *DepartmentResult* represents the result of the game player's decision in the department level. Example semantics of *DepartmentResult* is shown in the following figure.

DepartmentResult::getAchieved(): integer
precondition:
the number of professions whose productivity are achieved is defined

DepartmentResult::setExceedsBudget()
precondition:
this.exceedsBudget = false
prostondition:
this.exceedsBudget = true

…
// additional operations

Figure 5.23 Specification of the Interface of *DepartmentResult*

A department result has three variables: *achievedProfessions*, representing the number of professions that exist in the department whose productivity is achieved, *notAchievedProfessions*, representing the number of professions that exist in the department whose productivity is not achieved, and *exceedsBudget*, representing the value of whether or not the cost spent based on the game player's decisions exceed the budget allocated to the department. Operation *getAchieved* returns the number of professions whose productivity is achieved. Operation *setExceedsBudget* set the *exceedsBudget* variable to *true*, which means that the player's decisions results in costs that exceed the budget.

BusinessRoleResult

The architecture component *BusinessRoleResult* represents the result of the game player's decision for a certain profession. Example semantics of *BusinessRoleResult* is shown in the following figure.

```
BusinessRoleResult::setNewObsolete(n:integer)
postcondition:
        this.getNewObsolete = n

BusinessRoleResult::setProductivity(n:double)
postcondition:
        this.getProductivity = n

BusinessRoleResult::setObjectiveReached()
precondition:
        this.objectiveReached = false
postcondition:
        this. objectiveReached = true

…
// additional operations
```

Figure 5.24 Specification of the Interface of *Profession Result*

A profession result has *newObsolete* variable that represents the number of new obsolete people. Operation *setNewObsolete* sets the number of the new obsolete people to the given value. Another variable of profession result is *productivity* that represents the achieved productivity. Operation *setProductivity* sets the productivity of the profession to the given value. Variable *objectiveReached* describes whether or not the achieved productivity reaches the objective. Operation *setObjectiveReached* set the *objectiveReached* variable to *true*, which means that the productivity of the profession has reached the objective.

Finance

The architecture component *Finance* represents the cost spent on the profession and department level. Example semantics of *Finance* is shown in the following figure.

DepartmentResult::getFTRCost(): integer
precondition:
the FTR cost spent is already defined

DepartmentResult::getTotalCost(): integer
precondition:
the total cost spent for the entire profession or department
…
// additional operations

Figure 5.25 Specification of the Interface of *Finance*

A department result has variable *FTRCost* that represents the cost spent to pay all of the FTR employees that exist in the profession or the department. Operation *getFTRCost* returns the value of the *FTRCost* variable. The *totalCost* represents the total cost spent on employees exist in the entire profession or department. Operation *getTotalCost* variable returns the value of the *totalCost* variable.

BusinessRules

The architecture component *BusinessRules* represents the business rules that are applied to the department based on the game player's decision. The semantics of *BusinessRules* is shown in the following figure.

BusinessRules::calculateConfiguration(br:BusinessRole)
postcondition:
the new configuration of employee in the profession is calculated and set as the result of the profession

BusinessRules::calculateEffects(period:Period, br:BusinessRole)
postcondition:
the other effects on the profession level is calculated and set as the result of the profession

BusinessRules::calculateCost(dept:Department,bs:BusinessRole)
postcondition:
the cost spent for the profession is calculated and set as the finance of the profession

BusinessRules::calculateAchievedProd (dept:Department)
postcondition:
the number of professions that achieve and don't achieve their productivities are calculated and set as the result of the department

BusinessRules::calculateNewValues(game:Game)
postcondition:
the new values for the next period is calculated and set to the next game period

Figure 5.26 Specification of the Interface of *BusinessRules*

Operation *calculateConfiguration* calculates the new configuration of the given profession based directly on the game player's decision, creates *BusinessRoleResult* with those values and set it to the *BusinessRole*. Operation *calculateEffects* calculates the other effects that are not directly related to player's decision on reconfiguring the number of employees, whether it is because of changing a certain parameter in the department level or a disaster scenario that are applied to the current game period. The operation then sets those values to the *BusinessRoleResult* of the *BusinessRole*. Operation *calculateAchievedProd* calculates the number of professions that achieved their productivities and

those who don't, then creates *DepartmentResult* with those values and set it to the *Department*. Operation *calculateNewValues* calculates the new values on the department and profession level for the next game period and sets those values to the next *Period*.

Department

The architecture component *Department* represents the department that is played in the game. Example semantics of *Department* is shown in the following figure.

```
Department::addProfession(prof:BusinessRole)
postcondition:
        this.professions.get(n) = prof

Department::setDecision(dd:DepartmentDecision)
postcondition:
        this.decision = dd

Department::setFinance(f:Finance)
postcondition:
        this.finance = f

Department::setResult(dr:DepartmentResult)
postcondition:
        this.result = dr

…
// additional operations
```

Figure 5.27 Specification of the Interface of *Department*

A department has *professions* variable that represents array of professions exist in the department. Operation *addProfession* adds a given profession to the array of professions. The *decision* variable represents the decisions that are made by the game player in the department level. Operation *setDecision* sets the value of the *decision* to the given value. A department has *finance* variables represents the cost spent in the department level as the consequences of the decisions. The *result* variable represents the result of the decisions on the number of professions that achieved their objectives and those who are not. Operations *setFinance* and *setResult* respectively sets the *finance* and *result* variables to the given values.

BusinessRole

The architecture component *BusinessRole* represents each game period that is played in the game. Example semantics of *BusinessRole* is shown in the following figure.

BusinessRole::setDecision(brd:BusinessRoleDecision)
postcondition:
this.decision = brd

Department::setFinance(f:Finance)
postcondition:
this.finance = f

Department::setResult(brr:BusinessRoleResult)
postcondition:
this.result = brr

…
// additional operations

Figure 5.28 Specification of the Interface of *BusinessRole*

As a department, a business role has variables *decision, finance,* and *result* that represent the decisions, the cost spent, and the result of the decisions that are made in the profession level. Operations *setDecision, setFinance* and *setResult* respectively set the *decision, finance* and *result* variables to the given values.

Overall Interface Specification

To see how the interfaces described above take place in the architecture design, the components that take part in the Game Playing phase are shown again in Figure 5.29.

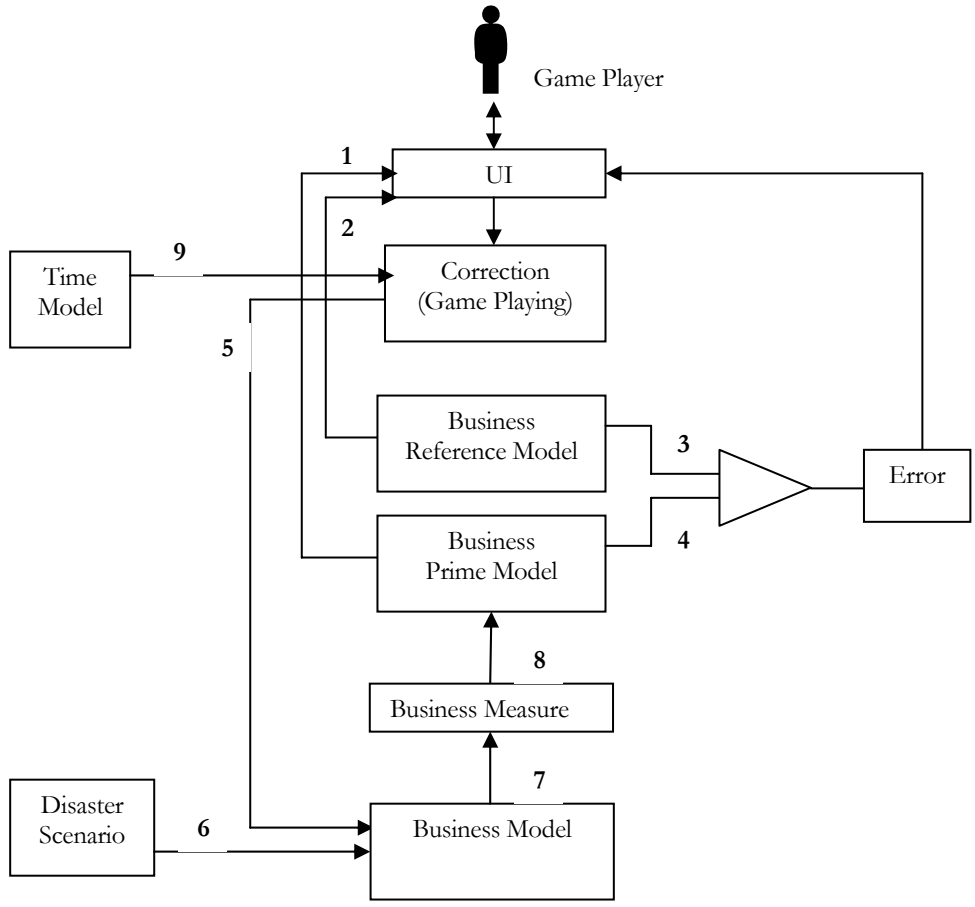

Figure 5.29 Components of the Game Playing Phase

The number shown in Figure 5.29 indicates the interface number to help describing each interface. By referring to the above figure, example of the overall Java interface specification of the game setup phase is described in Table 5.9.

Table 5.9 Interface Specification of the Game Playing Phase

Interface	Java Interface Specification
Interface 1	1. Game.createDepartment(**Department** department) 2. Game.createBusinessRole(**BusinessRole** profession) 3. DepartmentResult.getAchieved(): **int** 4. DepartmentResult.getNotAchieved() : **int** 5. DepartmentResult.isBudgetExceeded(): **Boolean**
Interface 2	1. Game.createStrategies(**BusinessStrategies** strategies)
Interface 3	1. BusinessRole.getObjective(): **double**
Interface 4	1. BusinessRole.getProductivity(): **double** 2. BusinessRoleResult.getProductivity(): **double**
Interface 5	1. Game.createDeptDecision(**int** n1, **int** n2) 2. Department.setDecision(**DepartementDecision** deptDecision) 3. BusinessRole.setDecision(**BusinessRoleDecision** profDecision)
Interface 6	1. Game.createDisaster(**Disasters** disasters) 2. Period.setDisaster(**String[]** dis)
Interface 7	1. DepartmentDecision.getEduDays() 2. DepartmentDecision.getRetDays() 3. BusinessRole.getNewObsolete()
Interface 8	1. BusinessRole.getProductivity() 2. Department.setResult()
Interface 9	1. Game.incrementIndex()

5.6 Dynamic Behavior of the Architecture

In order to model the dynamic behavior of the architecture, collaboration diagrams are used. To make a complete and understandable collaboration diagram, based on the interface specification described in the previous section. Therefore, the relationship and message flows among these components can be clarified. The collaboration diagrams are described for the game setup phase and the game playing phase.

5.6.1 Game Setup Phase

The collaboration diagram for the game setup phase of the SHAPE Workforce Management Game is depicted in Figure 5.30.

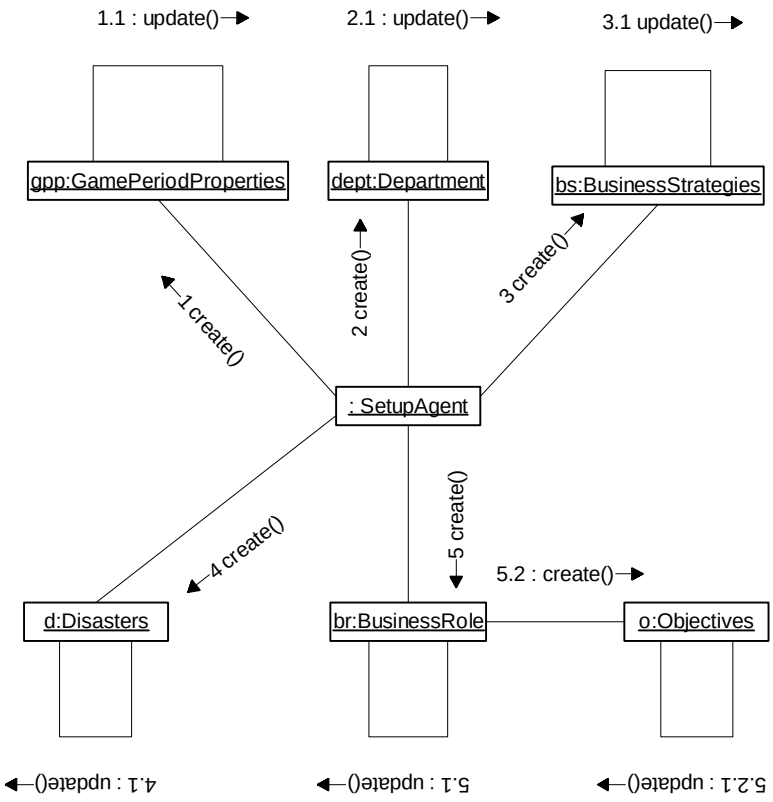

Figure 5.30 Collaboration Diagram of Game Setup

Figure 5.30 shows the flow of control involved in setting up the game environment during the game setup phase. The *SetupAgent* creates the five objects that will be involved in the game: a *GamePeriodProperties*, a *Department*, a *BusinessStrategies*, a *Disasters*, and a *BusinessRole.* Each of the objects will invoke the *update* operation on itself whenever a change is made to one of its values. The *BusinessRole* object will also create an *Objectives* object that represents the objective of the business role for each game period. The *Objectives* objects also will invoke the *update* operation on itself whenever a change is made to one of its values.

5.6.2 Game Playing Phase

The collaboration diagram for the game playing phase of the SHAPE Workforce Management Game is depicted in Figure 5.31.

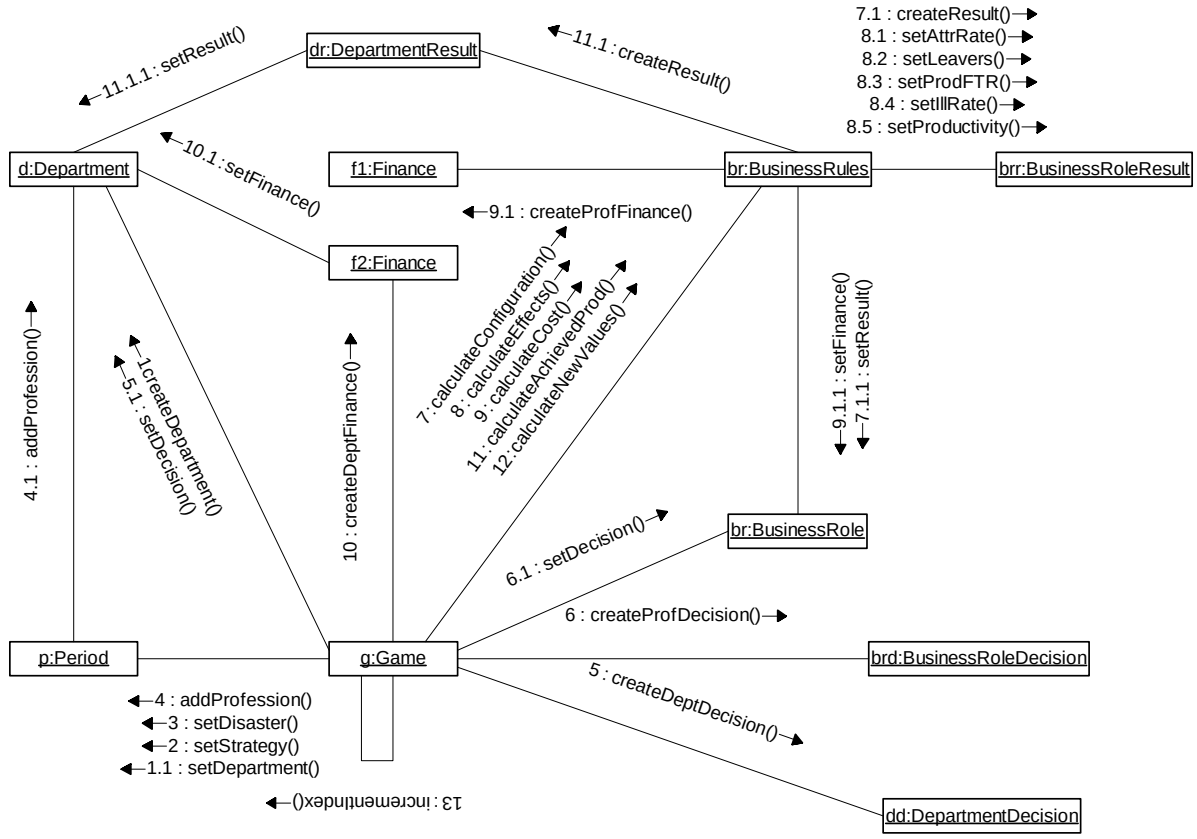

Figure 5.31 Collaboration Diagram for Game Play

There are ten objects shown in Figure 5.31: a *Game*, a *Period*, a *Department*, a *BusinessRole*, a *DepartmentDecision*, a *BusinessRoleDecision*, a *BusinessRules*, a *DepartmentResult*, a *BusinessRoleResult*, and a *Finance*. It specifies the flow of control involved in making decisions in department level, making decisions on one business role in that department, calculate all the effects of those decisions, and then go to the next game period. The action begins with the *Game* creating a *Department* object and setting the department to the *Period*. The game will also set the strategy and the disaster to the period. The game then adds the business role that exists in that period and the period adds the business role to the department where the profession belongs to. During the game playing, the game player makes decisions in the department level. The game creates a *DepartmentDecision* object that contains the decisions of the game player and sets the decision to the department. The game player also makes decisions on a *BusinessRole* that exists in that department. To store these decisions, the game creates a *BusinessRoleDecision* object and then sets the decision to the profession. After the game player decides to commit the decisions as final decisions, the game invokes first

calculateConfiguration on *BusinessRules* object who calculates the new configuration of the business role based straight on the game player's decision. Based on the result of the calculation, a *BusinessRoleResult* object is created and this result is set to the business role. The next operation of the business rules invoked by the game is *calculateEffects*, who calculates other effects that are not caused directly by the game player's decisions. Based on the result of the calculation, the business rules will then set the new attrition rate (*setAttrRate*), the number of other leavers besides those employees who are fired (*setLeavers*), the new number of productive FTR (*setProdFTR*), the new illness rate (*setIllRate*), and the current productivity (*setProductivity*) to the business role result. The third operation of the business rules is *calculateCost* who calculates the cost spent for that business role based on the player's decision and other effects. A *Finance* object is created to store the details of the cost spent and then set to the business role. After calculating the cost spent for all business roles, the game creates another *Finance* object to store the total cost spent for all business roles. In the example of Figure 5.31, since the player only make decisions one business role in the department, the game creates the *Finance* object after calculating the cost for that business role and then set it to the department. The game continues with invoking *calculatingAchievedProd* operation which calculates the number of business roles in the department that achieved their productivity and those who did not. The number calculated is used to create the *DepartmentResult* object which is then set to the department. The last business rules operation that is invoked is *calculateNewValues* that calculates the new values of department and business role properties that will be applied in the next period. After setting these new values, the game then go to the next period first by invoking *incrementIndex* operation on itself. The similar flow will then repeated for the next game period with the new values acquired from the previous one.

Chapter 6

EVALUATION OF ARCHITECTURE

6.1 Introduction

The architecture of SHAPE Workforce Management Game is evaluated in this chapter. The evaluation is a quality assessment process. The quality assessment is made up by several aspects, like robustness, reusability, adaptability, flexibility and security etc. This chapter will describe the architecture-level analysis that focuses on the stability and reusability aspects of the architecture. By comparing with the quality demand on those aspects, this chapter will show why the architecture is considered a stable architecture. Section 6.2 will describe the method that is used to evaluate the architecture and section 6.3 will describe the evaluation of the architecture.

6.2 Software Architecture Evaluation Method

The method that is used in this project to analyze the software architecture of this project consists of four steps. The first step is to develop several evolution scenarios. An evolution scenario is a description of some anticipated use of a system in the future. Each of these scenarios is then evaluated to see what kind of changes required for the system to perform the scenario. The third step is to find out which scenarios require changes in the same component of the system. This step is done to measure the level of separation of concerns supported by the architecture. Then the last step is to make the overall evaluation on the stability and reusability of the architecture.

6.3 Evaluation on SHAPE WMG Architecture

This section describes the analysis of SHAPE WMG Architecture using the method described previously. As explained before, there are four steps involved in the evaluation process. Each of these steps in the context of SHAPE WMG will be explained from section 6.3.1 to section 6.3.4.

6.3.1 Scenarios Development

The first step in the evaluation of the architecture design is to develop several evolution scenarios for different stakeholders. The scenarios should illustrate the kinds of activities the system must support and the kind of changes must be made to the system. In this project, four types of stakeholders are considered: *users*, *maintainers, system administrator*, and *developers*. Every type of these stakeholders has different concerns for the system. The users concern with the functionality and usability of the system, the maintainers concern with the maintainability and the ability to locate places of changes, the system administrators concern with the ease in finding source of operational

problems, and the developers concern with the clarity and the completeness of the architecture. The scenarios for each of the stakeholders involved in this project are described as follows:

Users (game managers):

1. Add more parameters for disaster scenarios
2. Generate random disaster scenarios
3. Include a profession from a certain period
4. Add other parameters to represent objective
5. Change the business rules

Users (game players):

6. Print out the game playing results
7. Record game playing steps and replay the game
8. Have a scoring system to indicate success or failure

Maintainers:

9. Make modifications to the user interface
10. Connect the game to external storage or printer
11. Prevent game user for giving input to unreasonable data

Administrators:

12. Change access permissions in the setup phase
13. Add function to recover from system failure

Developers:

14. Add artificial intelligence to support decision making process
15. Support multiple players playing at the same time as competitors

6.3.2 Evaluation on Evolution Scenarios

Every evolution scenario described in the previous section is then evaluated based on the kind of changes that is required for the scenario to be supported by the system. The changes are classified into one of the following categories:

A. No change required
B. Interface boundary is the same, new operation is required
C. Interface boundary is the same, change required in the implementation
D. Interface boundary is the same, change required in the function
E. Change required in the interface boundary

The evaluation for all of the scenarios based on the above categorization is shown in Table 6.1.

Table 6.1 Scenario Evaluation

Scenario	Description	Type of Change	Changes Required
1	Add more parameters for disaster scenarios	C	No additional component required, the changes should be made in the implementation level to determine how the disasters influence the application of the business rules
2	Generate random disaster scenarios	C	No additional component required, the changes should be made in the implementation of disaster scenario component that can also functions as a random generator
3	Include a profession from a certain period	A	
4	Add other parameters to represent objective	D	No additional component required, the changes should be made in the internal architecture of the business reference model for example by adding a new class to represent the objective
5	Change the business rules	B	No additional component required, the changes should be made in the class business rules for example by adding new operations
6	Print out the game playing results	A	
7	Record game playing steps and replay the game	D	No new component required, the changes should be made in changing the function of the game player modeling where the game player measurer can record the game player state and give these steps as input to business model to be replayed.
8	Have a scoring system to indicate success or failure	B	No additional component required, the changes should be made by adding functions to score the results, for example by creating a new class in the error component
9	Make modifications to the user interface	A	
10	Connect the game to external storage or printer	A	
11	Prevent game user for giving input to unreasonable data	B	No additional component required, the changes should be made by adding validation function when receiving input from the user
12	Change access permissions in the setup phase	A	
13	Add function to recover from system failure	E	A new component should be added, for example recovery manager, to guarantee that the system save the data periodically so when a system failure happen, several updates were already saved and can be loaded
14	Add artificial intelligence to support decision making process	D	No additional component required, the changes should be made in the functions of the components, for example the correction or the evaluation component
15	Support multiple players playing at the same time as competitors	E	A new component should be added to receive input results from other players and compare it with the local player's result, also other component to handle the concurrency control

6.3.3 Evaluation on Changes Required

From the scenarios evaluation described previously, it is shown that several scenarios require changes in several components or changes in the interface boundary. For the scenarios that require no changes in the interface boundary, the architecture is then considered to be stable since the architecture still can be used without having to make a severe change or add new component to the architecture. For the scenarios that require changes in the interface boundary, further evaluation should be made on whether or not the changes will affect the architecture that can result in restructuring the architecture to adjust to the changes.

The first scenario that needs changes in the interface boundary is evolution scenario 13 to add function to recover for system failure. This scenario will require a change in adding a recovery manager to the architecture. This change will not give a big impact on the current architecture. A recovery manager component can just be added into the architecture who receives and saves input from the game player and game manager user interface periodically, so that when a failure happens, the recovery manager can just refer to the last changes and upload it to the user interface.

The second scenario that needs to be evaluated further is evolution scenario 15 to support multiple players to play at the same time as competitors. This scenario will require additional comparator to compare the results and the decision made by the game players. The result could then be used as parameters in scoring the game. This change will not required restructuring the architecture, since the comparator can be added to connect the error components of each simulation modeling that belongs to each player and then add another component to represent the result of comparing the game player's error components.

6.3.4 Overall Evaluation

From the above evaluation, it can be seen that the current architecture is indeed stable and reusable for future development. From the fifteen evolution scenarios described previously, only two required changes in the interface boundary. It has been discussed that even with these scenarios the changes required are relatively small and don't require high cost and effort to add them. Although not all scenarios are covered in this evaluation step, it can be seen that the current architecture is stable enough and reusable for further development.

Chapter 7

IMPLEMENTATION AND TESTING

7.1 Introduction

The previous chapters describe the architecture design process, which is done based on the Synbad method. This chapter will describe the implementation of the architecture which results in a prototype of the SHAPE Workforce Management Game. A testing procedure is applied to test and validate the prototype in the final stage of this project. One of the important principles in developing software is that it should be maintainable. Maintenance is the most costly part in a software life cycle. In order to keep this prototype maintainable, the development tool that is used to implement SHAPE WMG is IBM WebSphere in Java on the Windows compatible platform. WebSphere masks the complexity of J2EE by allowing software developers to visually assemble applications from existing enterprise data. Its visual framework combines a J2EE development methodology with the ability to leverage existing and future application components making applications more maintainable and reusable.

The implementation consists of two important parts: the game setup part is focusing on the business model, which mainly covers the *Business Modeling* area shown in Figure 5.8. The game playing part mainly includes the business measurer, the business prime model, the business reference model, the correction, and the business model, which covers the *Simulation Modeling* area shown in Figure 5.8. The outline of this chapter is as follows. Section 7.2 gives an overview of the implementation of the game setup component. Section 7.3 gives an overview of the implementation of the game play component. Section 7.4 ends this chapter by describing the testing phase done for the implementation.

7.2 Game Setup Implementation

This section describes implementation of the game setup components depicted in Figure 5.8. These components are shown again in Figure 7.1.

Section 7.2.1 will describe the class diagram for the game setup components and section 7.2.2 will give the implementation of the classes in Java.

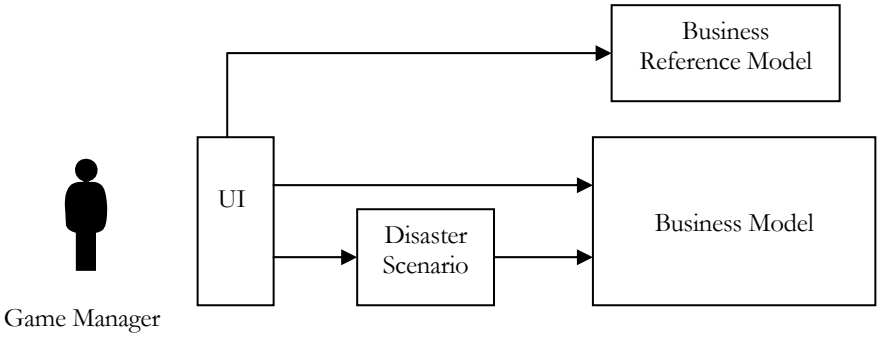

Figure 7.1 Components in the Game Setup Phase

7.2.1 Class Diagram

The class diagram depicted in Figure 7.2 describes the classes that are included in the three components shown in Figure 7.1 (excluding the *User Interface* component). Those components are: *Business Model*, *Business Reference Model*, and *Disaster Scenario*.

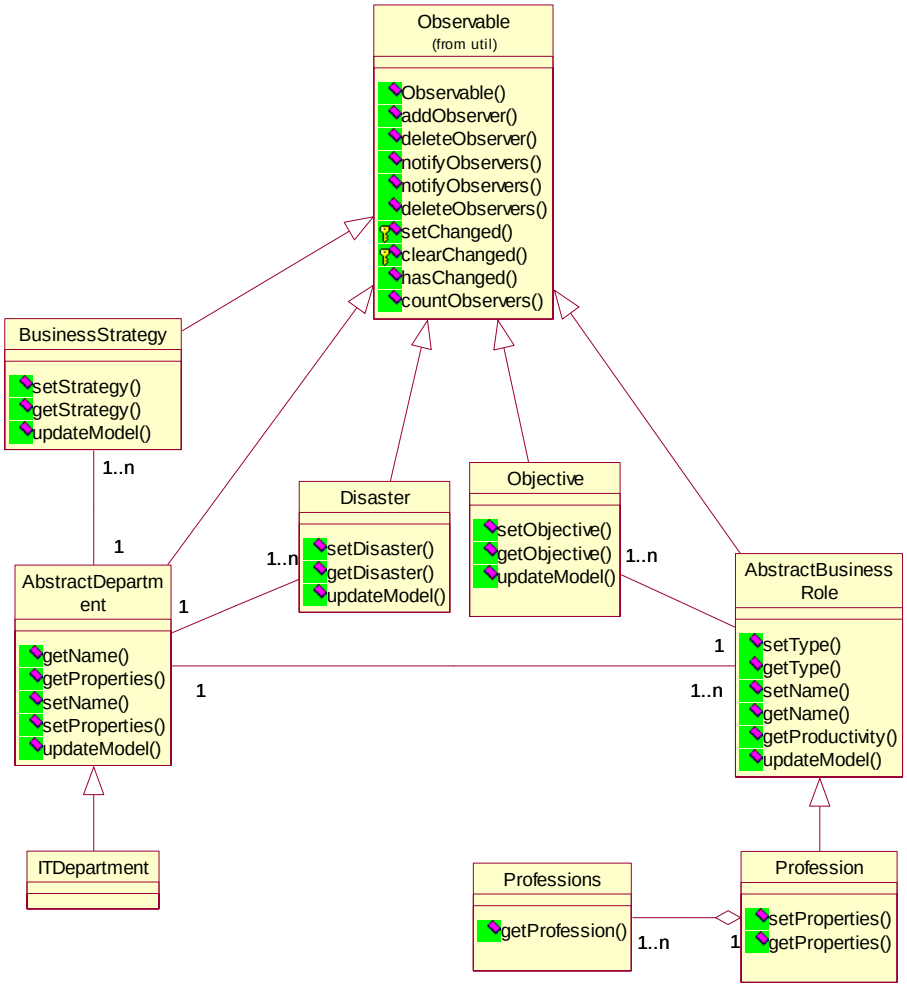

Figure 7.2 Class Diagram for Game Setup Phase Implementation

The *Abstract Department* class is an abstract class that is used as a foundation on which to build department classes by extension. This project focuses on the implementation of *IT Department* class which is implemented as the subclass of the *Abstract Department* class. This means that *IT Department* class inherits and implements the abstract method defined in *Abstract Department* class. Each department includes one or more business roles. As *Abstract Department* class, *Abstract Business Role* class is also an abstract class that is used as a foundation on which to build profession class. As already described, the subclasses for this class is the *Concrete Profession* class and the *Leaf Profession* class. In this project, the implementation is made only for the *Leaf Profession* class. This class is represented by the *Profession* class shown in Figure 7.2. The *Professions* class represents an array of *Profession* class, since a department can contain more than one profession.
Each profession has its own objectives. Therefore, the *Objective* class has association with the *Abstraction Business Role* class. The *Business Strategy* class represents the business strategy of a department for a certain period. Therefore, the *Abstraction Department* class can have one to many business strategies for one or more periods. The *Disaster* class gives exceptional effects on the properties of the IT department. All these classes extend the observable class. This will be explained more shortly. Some primary operations of each class are given in the class diagram.

Another game setup phase component that is shown in Figure 7.1 and has not been discussed yet is the *User Interface* component. To build the user interface in this phase the Model/View/Controller (MVC) pattern is used. In this pattern, the model component represents the classes shown in the class diagram described above, the view component represents the user interface component, and the controller component is the component that defines the way user interface reacts to user input. Another pattern that is applied to the *User Interface* component is the Observer pattern. The Observer pattern is used to model the objects that need to notify and update other objects when they change state. In this case, the Observer is the user interface which observes the state changes of its Subject. In Java term this Subject is called Observable. As already explained in the previous paragraph, the Observable subjects are the classes that extend the Observable class.
The implementation of the game setup components are decomposed into five Java packages: *BusinessModel, Controller, View, TimeModel,* and *Tools* packages. The diagram of these packages is shown in Figure 7.3.

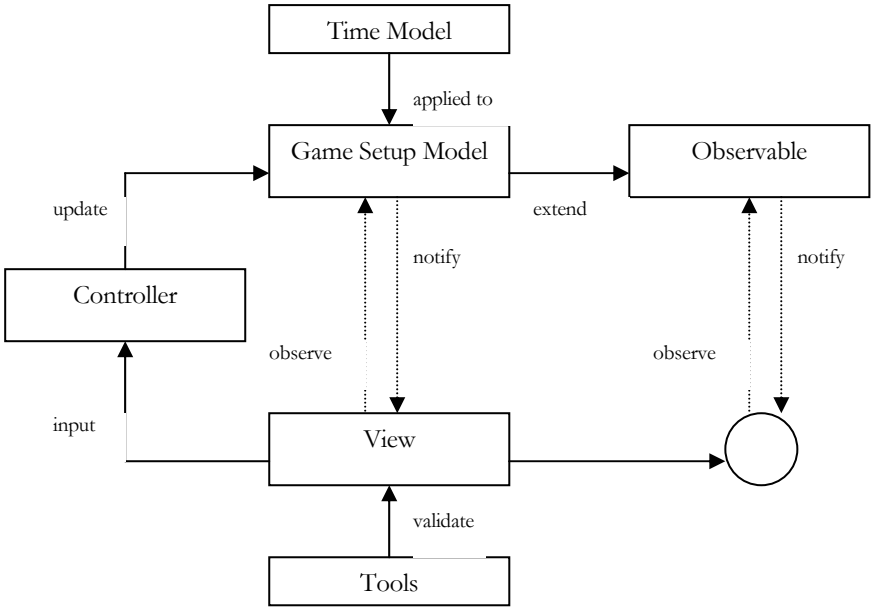

Figure 7.3 Package Diagram for Game Setup Phase Implementation

The *View* component determines how to display the model components. For every setup phase, there is a view component that displays all the required information to be filled in by the game manager. The class diagram for the *View* component is shown in Figure 7.4.

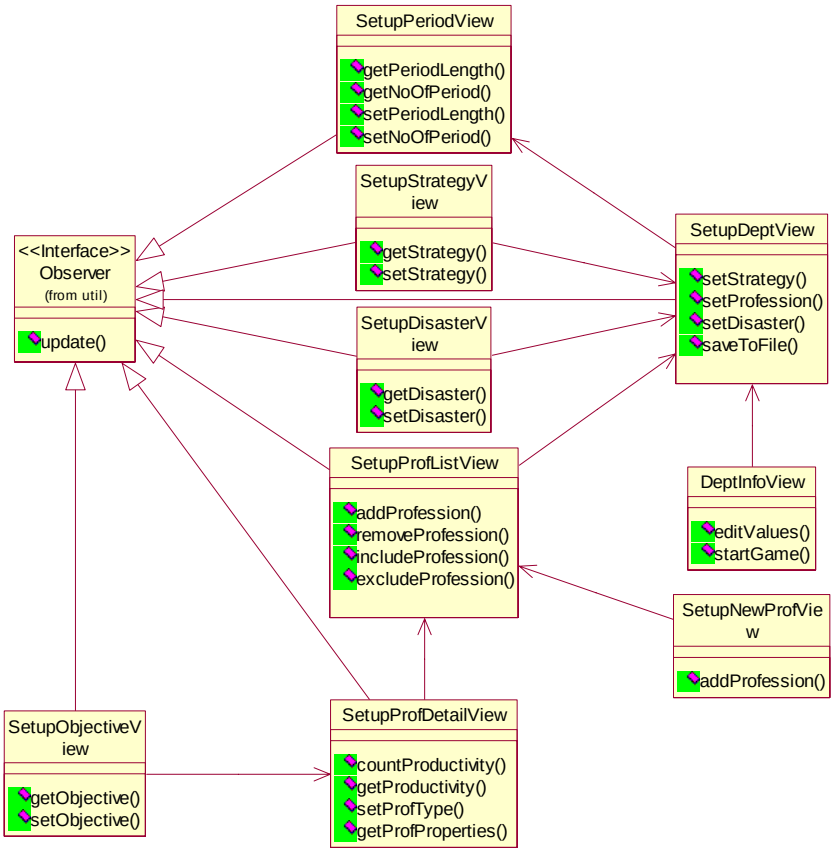

Figure 7.4 Class Diagram of View Component

All classes extends the JFrame class in order to provide a user friendly GUI. Those classes, except *DeptInfoView* class, also implement the Observer Interface to receive notifications. Basically *SetupDeptView* class has relation with the *SetupPeriodView* class, since *SetupPeriodView* creates the *SetupDeptView* and delivers the number of game period to the *SetupDeptView*. From the *SetupDeptView*, four other interfaces can be started: *SetupStrategyView*, *SetupDisasterView, SetupProfListView*, and *DeptInfoView*. So these four classes have relations with the *SetupDeptView*. The *SetupNewProfView* is started when the game manager wants to add a new profession to the list shown in the *SetupProfListView*. The *SetupProfDetailView* is also started when a certain profession is chosen from the list also shown in the *SetupProfListView*. From the *SetupProfDetailView*, *SetupObjectiveView* is started to set the objectives of the profession. The *DeptInfoView* displays all the values that have been set during the game setup phase, after saving the values into a file.

The *Controller* determines the actions that follow the game manager's input. The *Controller* modifies the *Game Setup Model* component. The class diagram for the *Controller* component is depicted in Figure 7.5.

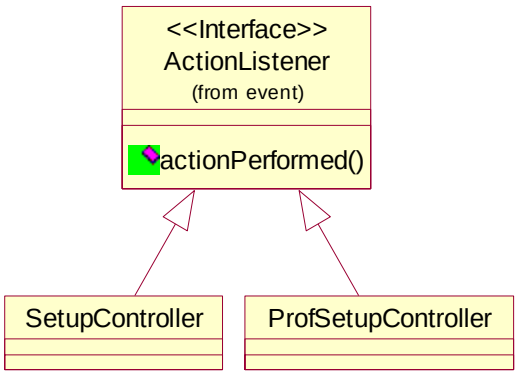

Figure 7.5 Class Diagram of Controller Component

The *ProfessionSetupController* class determines the actions that follow the game manager's input on the Profession List Setup (represented by *SetupProfListView* class) and Profession Details Setup (represented by *SetupProfDetailView* class) interfaces. This controller is separated by the other controller because it is more complex than the others. The controller for the rest of the interfaces in the game setup phase is combined in one class called *SetupController* class.

The class diagram of the Timing Controller package is shown in the following figure. This package only includes one class, *Game Period Properties*. It is used to set up the game time and give time controlling to the game play.

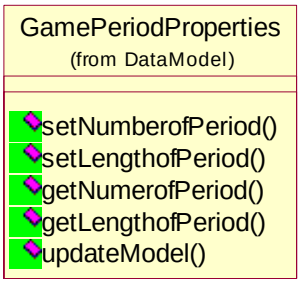

Figure 7.6 Class diagram of the Time Model Package

7.2.2 Coding of the Game Setup

According to the package diagram in Figure 7.3, five packages are defined within the project. Based on the class diagram described in the previous section, related classes are created those packages. The screen capture in Figure 7.7 shows the packages and classes organization in the project SHAPE_WMG.

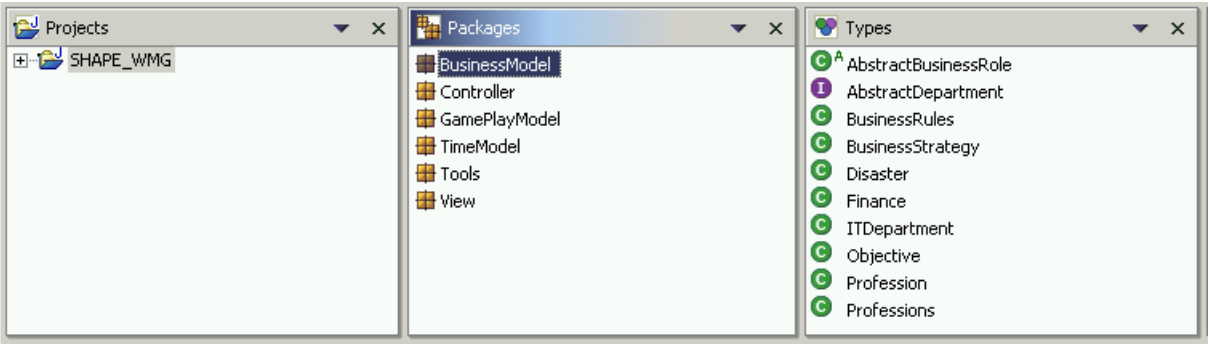

Figure 7.7 Packages and Classes Organization of the Project SHAPE_WMG

Remarks: the game play model is used for the game play phase. So the business model, controller, time model, tools and view are the five packages which are used to compose the game setup.

In order to explain the coding process, the IT department setup is given as an example. The table 7.1 presents the classes and their corresponding packages for the IT department setup.

Table 7.1 Classes and Packages for IT Department Setup

Class	Corresponding Package
ITDepartment.java	Business Model
SetupController.java	Controller
SetupDeptView.java	View

Parts of java codes of the above classes are given in the following Figure 7.8, Figure 7.9 and Figure 7.10.

ITDepartment.java | SetupController.java | SetupDeptView.java

```
package BusinessModel;

import java.util.Observable;
import java.util.Vector;
import java.io.Serializable;
import GamePlayModel.DeptDecision;

/**
 * @author Kevin He & Elisabeth Maya
 *
 */

public class ITDepartment extends Observable implements AbstractDepartment, Serial

    private int[] allprop = new int[11];
    private String name;
    private Vector professions = new Vector();
    private Finance finance;
    private DeptDecision decision;

    // CONSTRUCTOR

    public ITDepartment(){
        allprop[0] = 10;       // education days
        allprop[1] = 28;       // vacation days
        allprop[2] = 90;       // learning curve FTR
        allprop[3] = 21;       // learning curve FTS
        allprop[4] = 21;       // learning curve obsolete
        allprop[5] = 10000;    // education cost
        allprop[6] = 3000;     // retention cost
        allprop[7] = 5000;     // recruitment cost
        allprop[8] = 1000000; // total budget
        allprop[9] = 3;        // budget change
        allprop[10] = 50;      // golden handshake rate
    }

    // QUERIES

    public int[] getAllProp()
    { return allprop; }
```

Figure 7.8 Example Codes of ITDepartment.java

From the above figure, all the properties of the IT department and the constructor are shown. This class extends Observable class and implements the abstract department. So it inherits the properties and operations of the abstract department and can be observed by the view class. In order to save objects, this class implements the serializable interface. When any changes take place in this class, methods: *setChange () and notifyObservers ()* will be invoked to ask the user interface to refresh the display.

ITDepartment.java | SetupController.java | SetupDeptView.java

```
import BusinessModel.*;

/**
 * @author Kevin He & Elisabeth Maya
 *
 */

public class SetupController implements ActionListener {

    private SetupStarter ssView;
    private DeptInfoView diView;
    private SetupDeptView sdView;
    private SetupPeriodView spView;
    private SetupStrategyView strView;
    private SetupDisasterView disView;
    private WarningMessage wmView;
    private Disaster disaster;
    private ITDepartment department;
    private BusinessStrategy strategy;
    private GamePeriodProperties gamePeriod;

    public SetupController(SetupStarter ss)
    { ssView = ss; }

    public SetupController(DeptInfoView di)
    { diView = di; }

    public SetupController(GamePeriodProperties gpp, SetupPeriodView sp)  {
        gamePeriod = gpp;
        spView = sp;
    }

    public SetupController(WarningMessage wm)
    { wmView = wm; }

    public SetupController(ITDepartment dept, SetupDeptView sd)  {
        department = dept;
        sdView = sd;
    }
```

Figure 7.9 Example Codes of SetupController.java

This class implements ActionListener. This means that when a certain action happens in the user interface, for instance clicking a button, the SetupController.java captures this action and carries out the corresponding handling. There are five possible actions that could be done in the department setup user interface: *Back* (go back to the game period setup), *Strategy* (setup the business strategy for every game period), *Profession* (setup the professions that are going to be included in the game), *Disaster* (setup the disaster scenario to be applied in every game period), and *Finish* (complete the whole setup phase and save the setup values in a file).

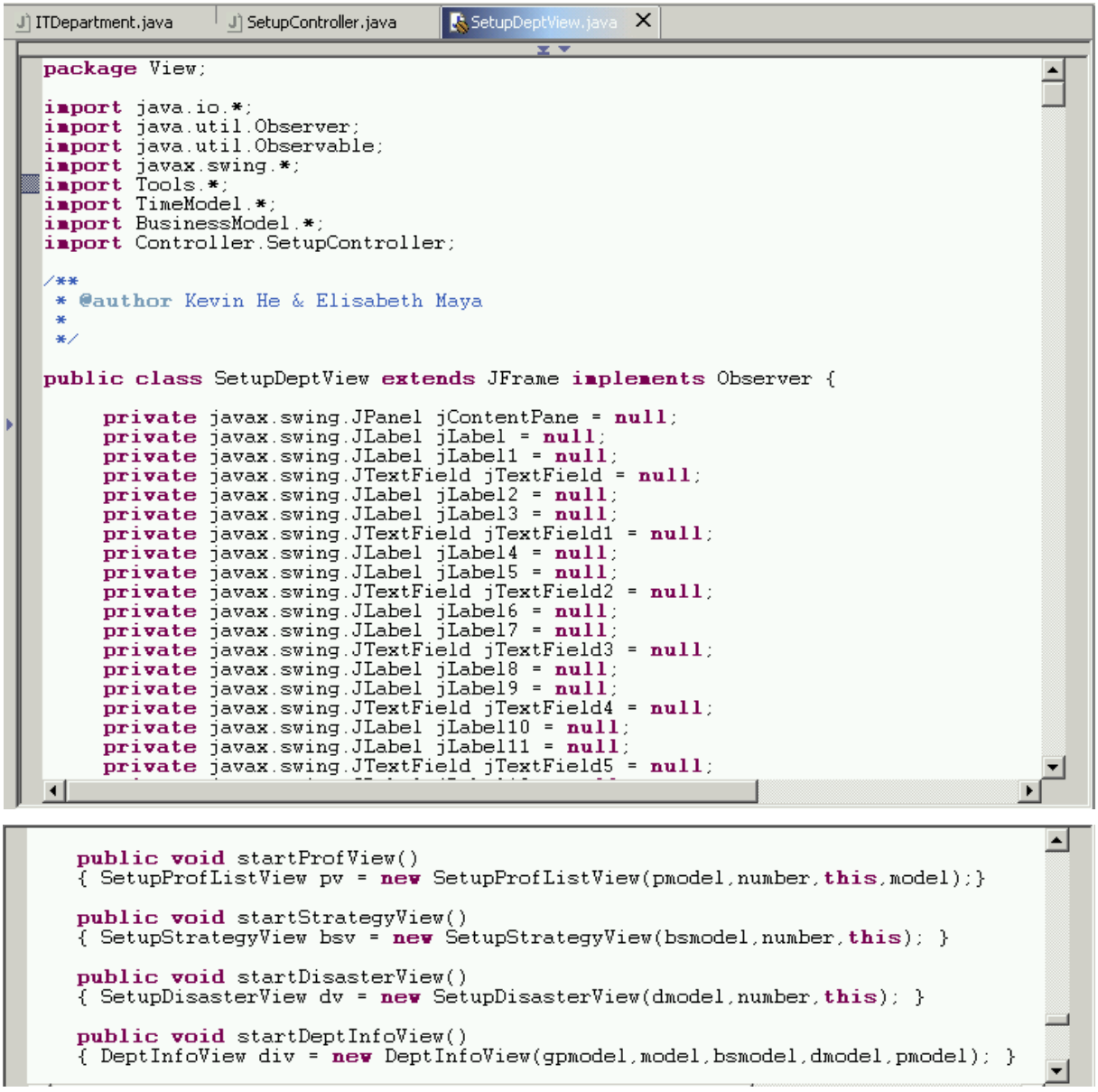

Figure 7.10 Example Codes of SetupDeptView.java

This class is actually used to create the user interface of the department setup. It extends the JFrame class to provide a user friendly GUI. It also extends the Observer in order to receive notification from the observable classes. The operations given in the above screen capture are used to start other setup user interfaces. The following Figure 7.11 is the screen capture of IT department setup interface.

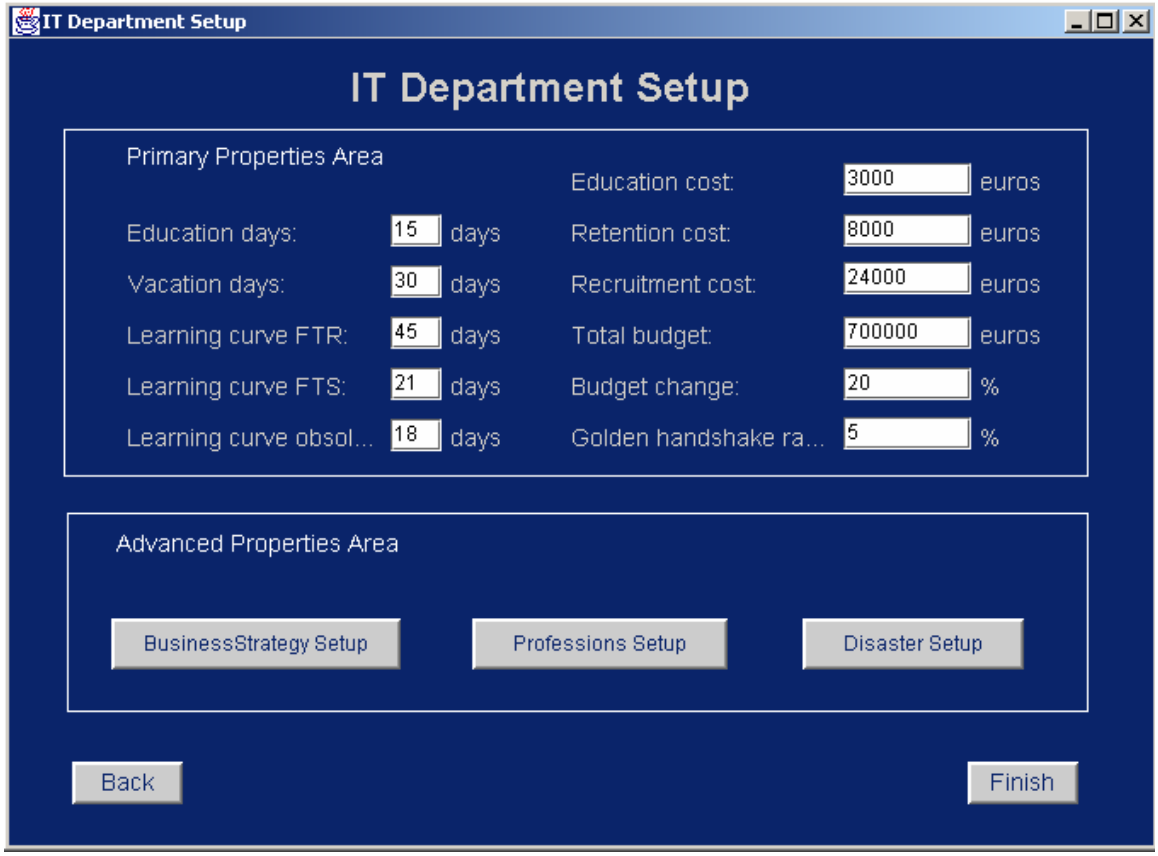

Figure 7.11 IT Department Setup Interface

7.3 Game Play Implementation

Referring to Figure 5.8, the components that take part in the game playing phase and are implemented in this project are shown again in Figure 7.12. As in the previous section, this section will first describe the class diagrams for all related components in the game playing phase. The class diagrams are also modeled using the Model-View-Controller pattern and Observable-Observer relation in UML. Then it is followed by the coding of the game playing phase in Java.

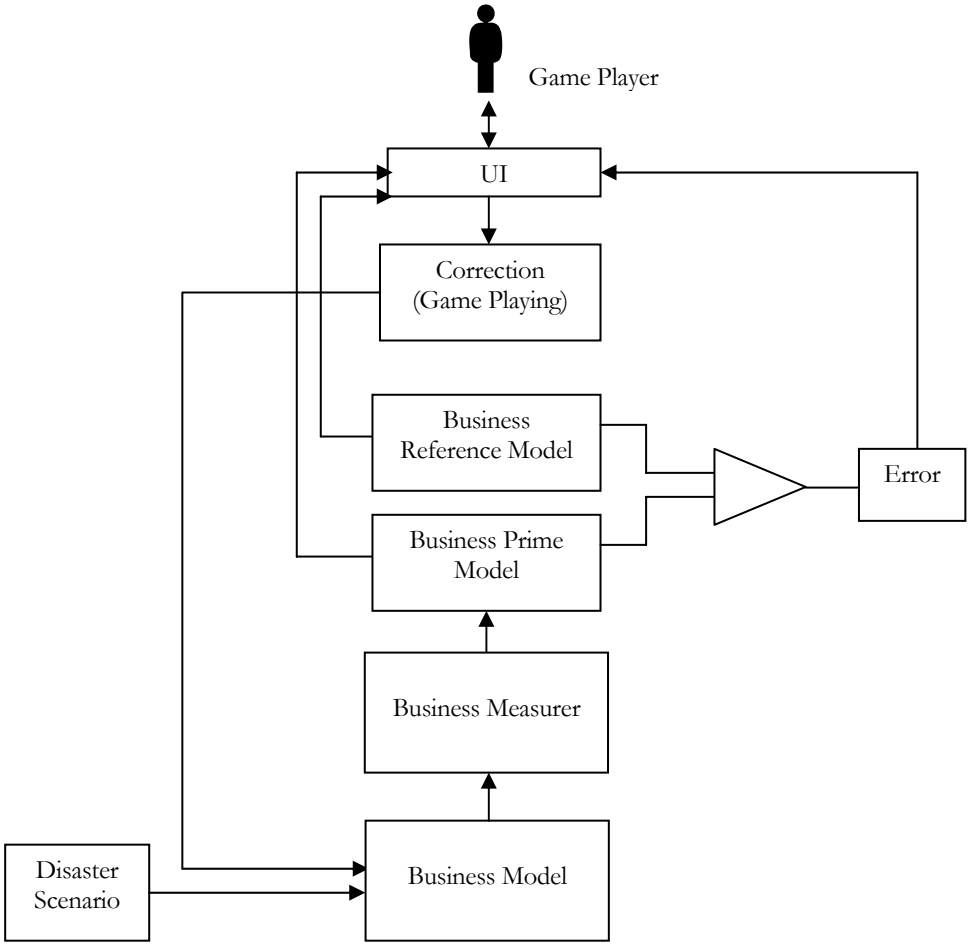

Figure 7.12 Components in the Game Playing Phase

7.3.1 Class Diagram

The class diagram depicted in Figure 7.13 describes the classes that are included in the seven components shown in Figure 7.12 (excluding *User Interface* component). Those components are: *Disaster Scenario, Business Model*, *Business Measurer, Business Prime Model, Business Reference Model*, *Error,* and *Correction*.

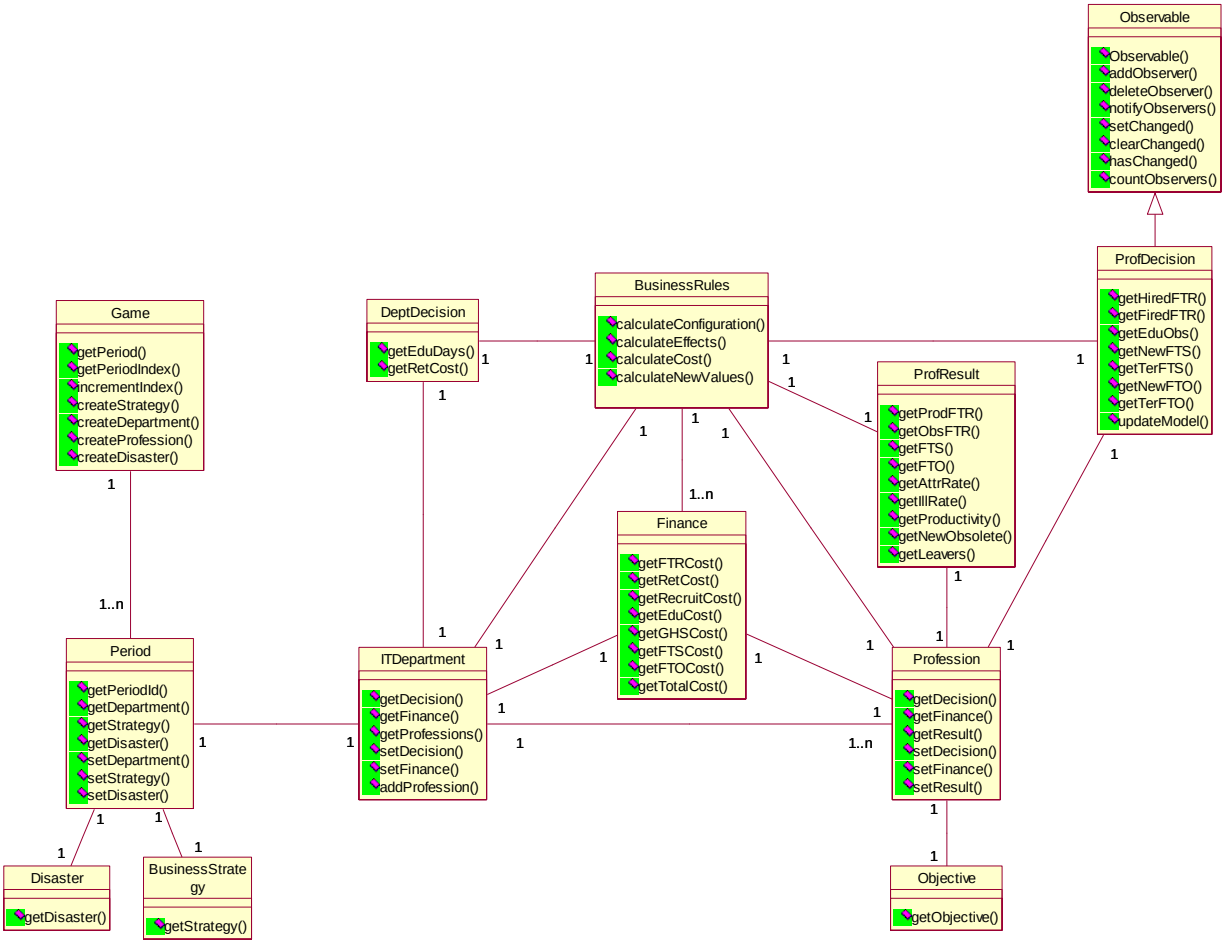

Figure 7.13 Class Diagram for Game Playing Phase Implementation

The *Game* class represents an instantiation of one game session. A *Game* can consists of one to many game periods. The *Period* class represents each of these game periods. Every *Period* has a *Disaster* and *Business Strategy* that has been set in the game setup phase. Every *Period* includes an *ITDepartment* in the game. An *ITDepartment* can have at least one *Profession* class. Every *Profession* has an objective for every period. This *Objective* is set in the game setup phase.

During a game period, a game player can make decisions on the department level or the profession level. These decisions are represented by *DeptDecision* and *ProfDecision* classes. The *ProfDecision* class extends *Observable* class. *BusinessRules* class represents the rules that are applied to the department and profession decisions. The result of applying these business rules are represented by two classes: *Finance* and *ProfResult*. The *Finance* class represents the detail and total cost spent in a certain period for every *Profession* and for the whole *IT Department*. The *ProfResult* class represents the result of the player's decisions on the current situation of every *Profession* in terms of employee configuration.

As in the game setup phase, the game playing phase also uses the Model/View/Controller (MVC) pattern to build the user interface.

The implementation of the game play components are decomposed into six Java packages: *BusinessModel, GamePlayModel, Controller, View, TimeModel,* and *Tools* packages. The diagram of the packages is shown in Figure 7.14.

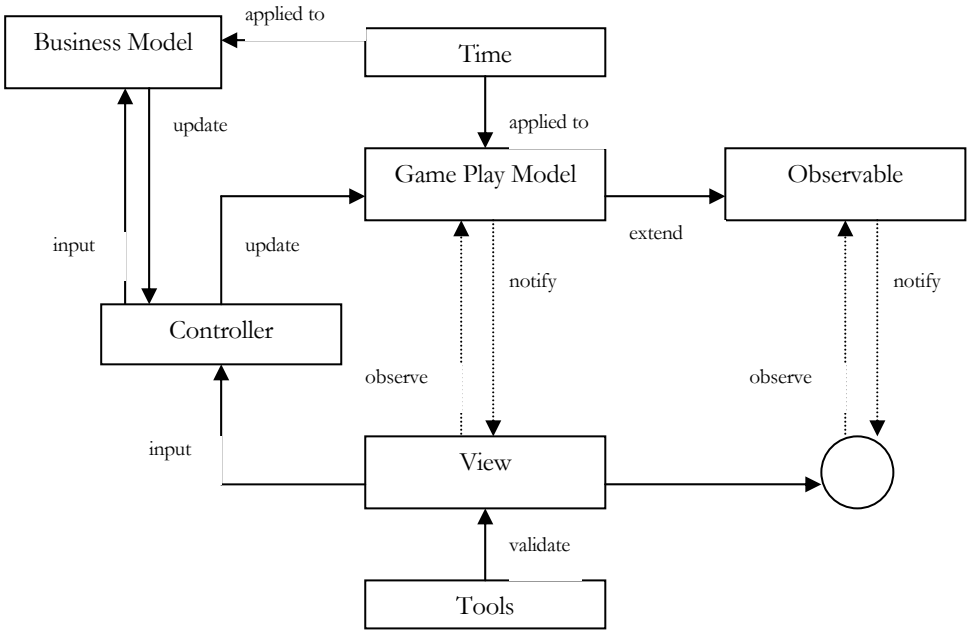

Figure 7.14 Package Diagram for Game Playing Phase Implementation

The View component determines how to display the model components. The class diagram for the View component is shown in Figure 7.15.

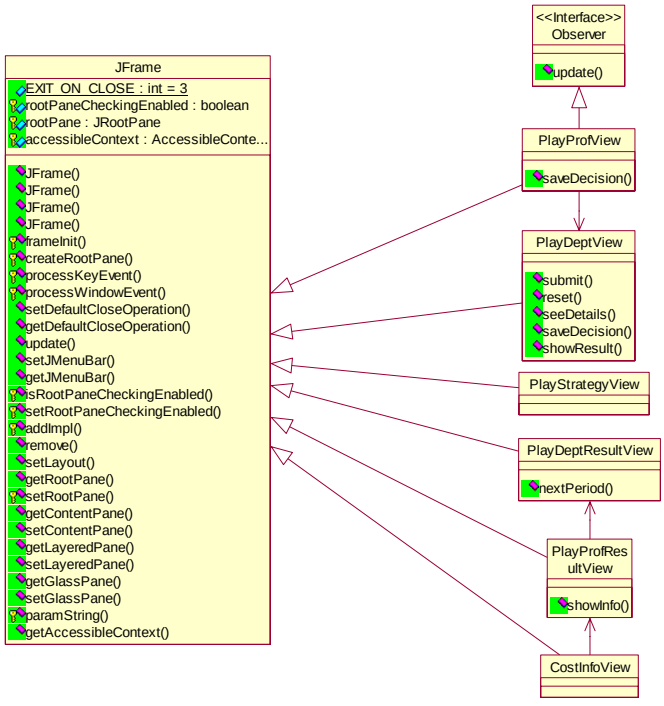

Figure 7.15 Class Diagram of View Component

All classes extends the JFrame class in order to provide a user friendly GUI. The first user interface shown to the game player is the *PlayStrategyView* class that displays the current game period number and the business strategy of that period. After this the game player is shown the current department situation through *PlayDeptView* class. *PlayDeptView* creates *PlayProfView* class when a certain profession is chosen from the list shown in *PlayDeptView*. *PlayProfView* class implements the Observer Interface to receive notifications. When the game player commits the final decisions, the results are then displayed. The first result shown is for the department level through *PlayDeptResultView* class. *PlayDeptResultView* creates *PlayProfResultView* class when a certain profession is chosen from the list shown in *PlayDeptResultView* to display the result in the profession level. The *CostInfoView* displays the information of cost spent per employee to help the game player in understanding the overall result of the profession.

The *Controller* determines the actions that follow the game player's input. It modifies the *Game Play Model* and *Business Model* component. The class diagram for the *Controller* component is depicted in Figure 7.16.

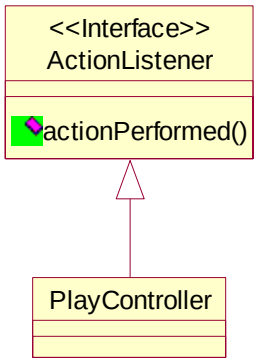

Figure 7.16 Class Diagram of Controller Component

The *PlayController* class determines the actions that follow the game player's input during the whole game playing phase. The actions that are done in different view components of the game playing phase are all handled by this one controller.

7.3.2 Coding of the Game Playing

According to the package diagram in Figure 7.14, six packages are defined within the project for the game playing phase. Based on the class diagram described in the previous section, related classes are created in those packages. The screen capture in Figure 7.17 shows the packages and classes organization in the project SHAPE_WMG.

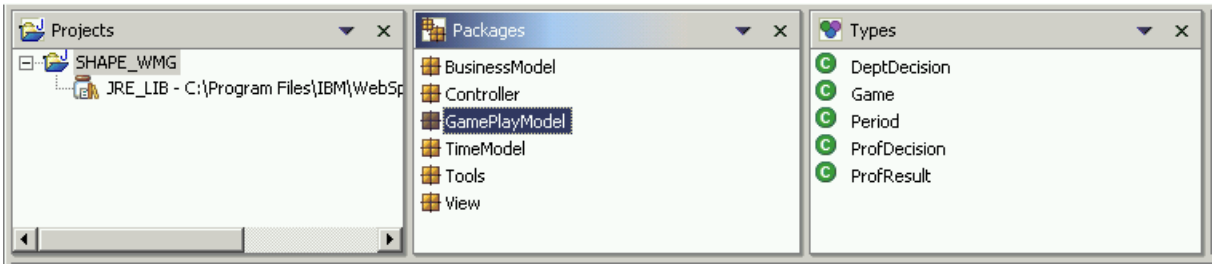

Figure 7.17 Packages and Classes Organization of the Project SHAPE_WMG

In order to explain the coding process, Table 7.2 presents the classes and their corresponding package for making decisions on the profession level.

Table 7.2 Classes and Packages for Making Decision on Profession Level

Class	Corresponding Package
ProfDecision.java	GamePlayModel
PlayController.java	Controller
PlayProfView.java	View

Parts of java codes of the above classes are given in the following Figure 7.18, Figure 7.19 and Figure 7.20.

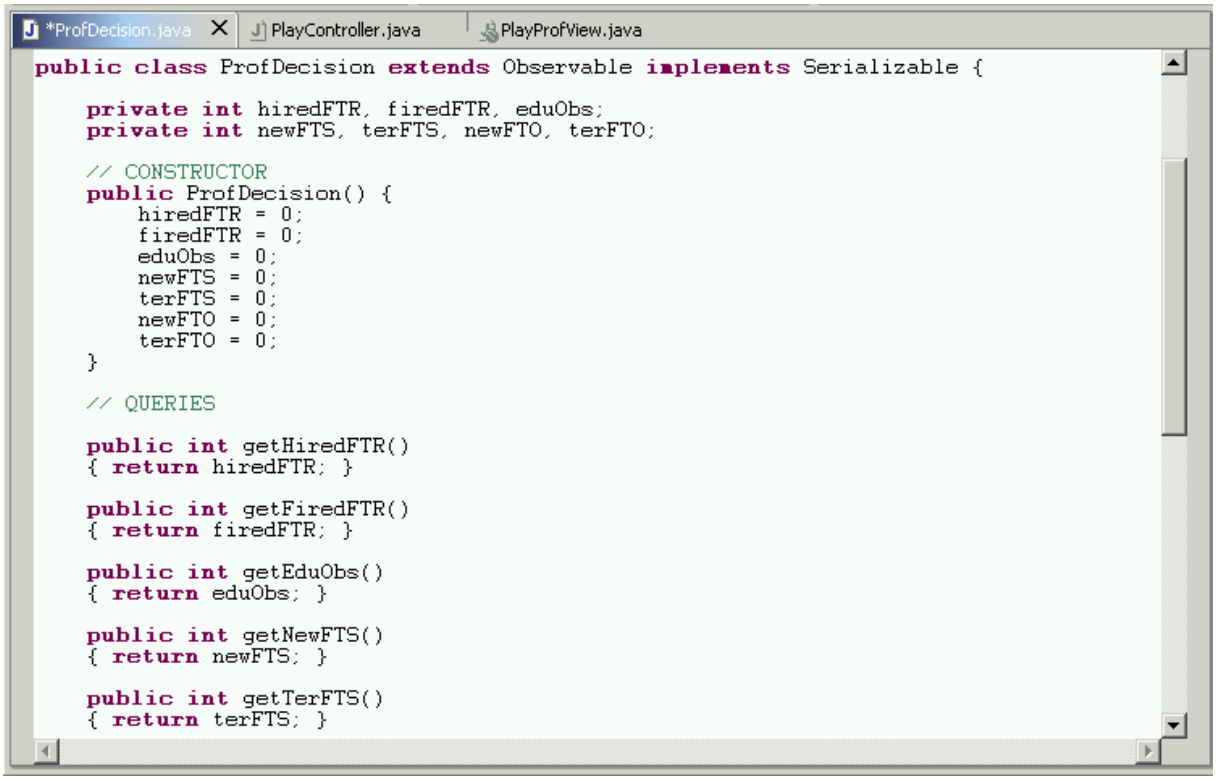

Figure 7.18 Example Codes of ProfDecision.java

Figure 7.18 shows the constructor of *ProfDecision* class. This class extends Observable class so it can be observed by the view class. In order to save objects, this class implements the Serializable interface. When any changes take place in this class, methods: *setChange () and notifyObservers ()* will be invoked to ask the user interface to refresh the display.

*ProfDecision.java | PlayController.java | PlayProfView.java

```
public class PlayController implements ActionListener {

    private PlayStarter psView;
    private PlayDeptView pdView;
    private PlayProfView ppView;
    private CostInfoView ciView;
    private PlayStrategyView spView;
    private PlayDeptResultView pdrView;
    private PlayProfResultView pprView;

    public PlayController(PlayStarter ps)
    { psView = ps; }

    public PlayController(PlayStrategyView sp)
    { spView = sp; }

    public PlayController(PlayDeptView pd)
    { pdView = pd; }

    public PlayController(PlayProfView pp)
    { ppView = pp; }

    public PlayController(PlayDeptResultView pdr)
    { pdrView = pdr; }

    public PlayController(PlayProfResultView ppr)
    { pprView = ppr; }

    public PlayController(CostInfoView ci)
    { ciView = ci; }

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (command.equals("NEXT1")) {
            psView.dispose();
            new PlayStrategyView(psView.getGame());
        }
        else if (command.equals("NEXT2")) {
            spView.dispose();
            new PlayDeptView(spView.getGame());
```

Figure 7.19 Example Codes of PlayController.java

Figure 7.19 show that the *PlayController* class implements *ActionListener*. Once certain actions happen in the user interface, for instance click button and press enter etc, the PlayController.java captures this action and carries out the corresponding handling. The PlayController class handles all actions that happen during the game playing phase.

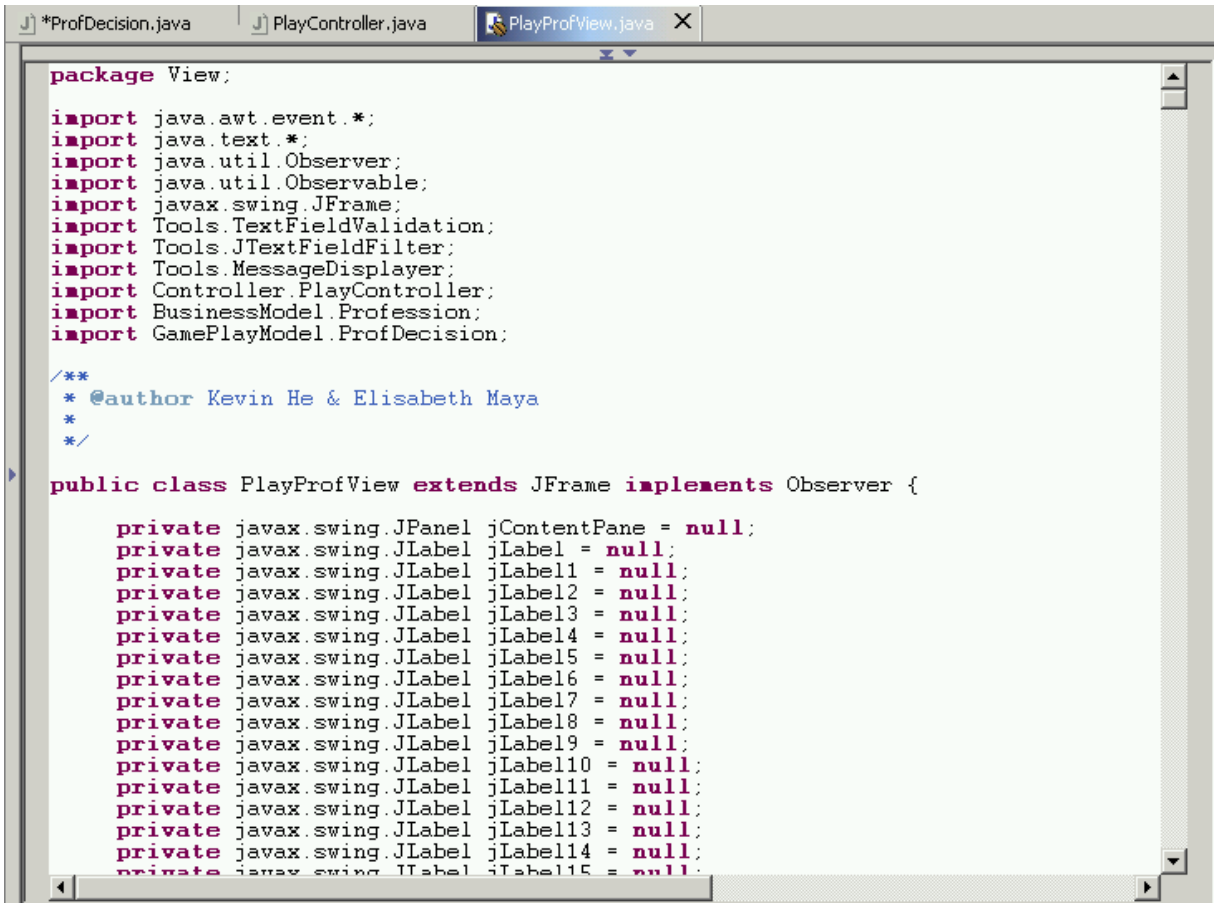

Figure 7.20 Example Codes of PlayProfView.java

The *PlayProfView* class is used to create the user interface for making decisions on a certain profession. It extends the *JFrame* class to provide a user friendly GUI. It also extends the *Observer* class in order to receive notification from the observable classes. The observable class of the *PlayProfView* is the *ProfDecision* class as seen in the following Figure 7.21.

```
public PlayProfView(Profession p,int n) {
    super("SHAPE WMG – Game Play");
    controller = new PlayController(this);
    profession = p;
    type = profession.getProfType();
    periodNo = n;
    initialize();
    ProfDecision decision = profession.getDecision();
    if (type != 1) {
        profession.getDecision().addObserver(this);
        update(profession.getDecision(),null);
    }
}
```

Figure 7.21 Constructor of *PlayProfView* class

Figure 7.21 show that *PlayProfView* class adds itself as the observer of the observable class *ProfDecision* which is invoked by the method:

profession.getDecision().addObserver(this)

The following Figure 7.22 is the screen capture for making decisions on the profession level

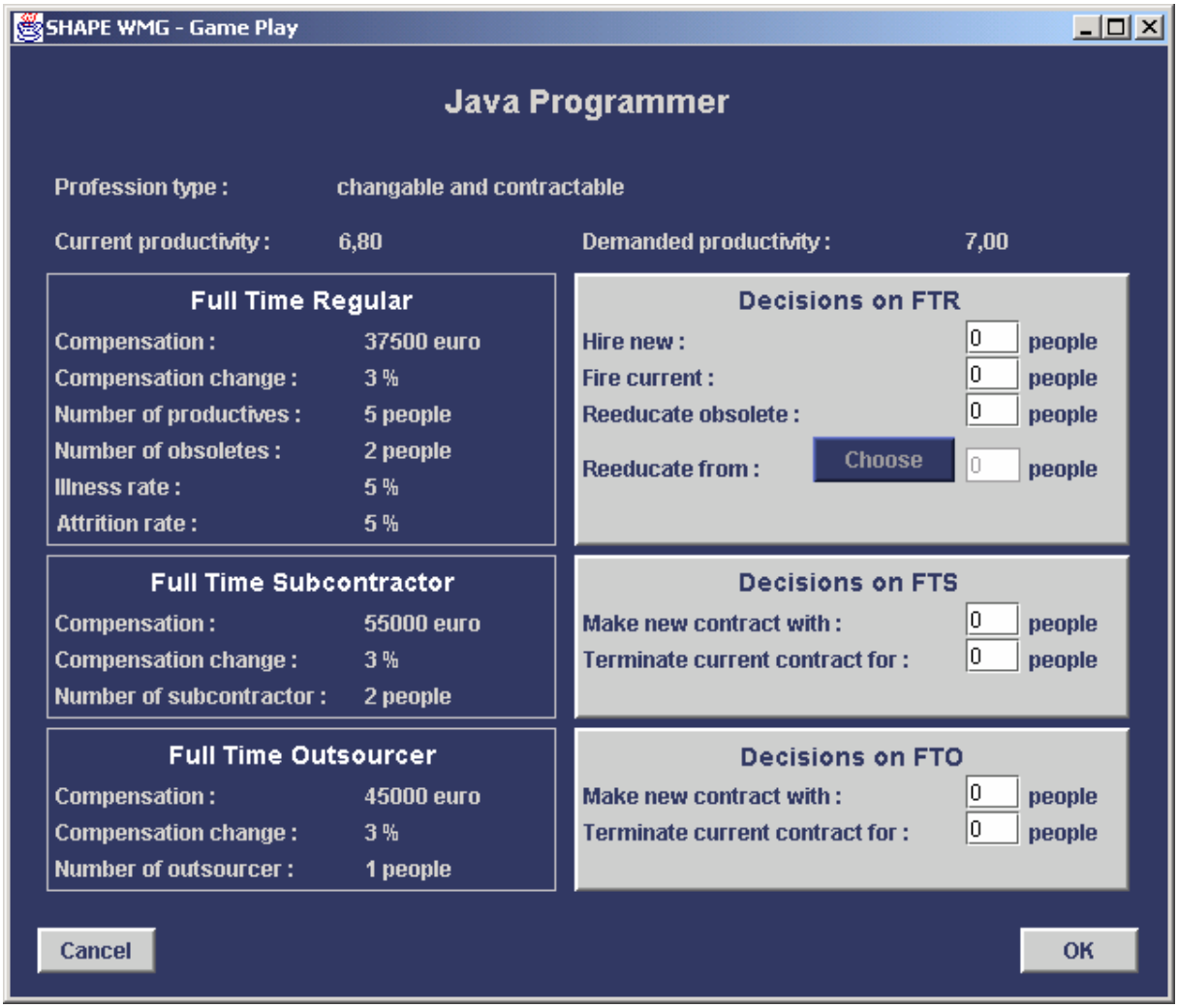

Figure 7.22 Profession Level Decision Making Interface

The left side in Figure 7.22 shows the current situation of the profession "Java Programmer". It shows the configuration of employees that are currently working as Java Programmers and also some properties related to the profession. The user interface also shows the current productivity of the profession. In the right side of the interface, it shows several decisions that can be made for this profession. The decisions made should be aiming to the demanded productivity that is shown on top of the possible decisions area.

7.4 Testing

The final acceptance test was done with IBM consultants in a workshop. The main goal is to verify that the prototype meets the requirements of SHAPE_WMG. Because making a complete and automated testing procedure for the prototype is currently not the high priority, a basic black-box test is carried out in a system test level. Other nonfunctional tests, for instance security testing,

performance testing, compatibility testing and usability testing are not yet considered in the current stage. [Lev02], [V2M2], [Bei90], [Bei95], [Rop94], [Sie96], [Hut03]

The testing procedure consists of two stages, game setup tests and game play tests. The test cases are basically made based on the use case scenarios, which are described in Chapter 3. By categorizing and prioritizing the test cases, three typical test cases are presented: import game setup file, make department decisions and make profession decisions. Other important test cases can be found in Appendix C. The terms used in the testing procedure are explained in the follows: [Dus03]

- ***Test Procedure ID***: the following naming conventions are used: S01 means the first test case of the game setup and P01 means the second test case of the game play test.
- ***Test Name***: a description of the test procedure.
- ***Test Procedure Author***: the developer of the test procedure.
- ***Test Objective***: the objective of the test procedure.
- ***Related use Case(s)***: provide the related use case scenario(s).
- ***Precondition/Assumption***: define the conditions, with which must be satisfied before executing the particular test procedure.
- ***Post condition***: define the conditions, with which must be satisfied after executing the particular test procedure.
- ***Verification Method***: include automated, certification, manual test or analysis.
- ***User Action (Inputs)***: the inputs, which are needed to create a test.
- ***Expected Results***: define the results expected when executing the particular test procedure.
- ***Trace Log Information***: document the behavior of back-end components. For example, write log entries detailing the functions they are executing and the major objects with which they are interacting. In our case, the log information is displayed in the DOS command window.
- ***Actual Result***: this field may have a default values, such as "Same as Expected Result", that is changed to describe that actual result if the test-procedure fails.
- ***Pass/Fail***: shows the test is passed or failed.
- ***Test Environment***: in which environment this test has been done. For example: the operation system and the computer equipments etc

7.4.1 Game Setup Test Cases

Table 7.3 shows the testing procedure of importing game setup file test case. This test case takes place in the game setup phase. In order to reuse the previous setup file, the user can search for a setup file and import it in the computer local disk. The setup file is identified with an ".obj" extension file. After this file is successfully imported, all setup values of this file will be used in the game.

Table 7.3 Testing Procedure of Importing Game Setup File

Test Procedure ID	S01
Test Name	Import game setup file
Test Procedure Author	Maya & Kevin
Test Objective	To test if the setup values can be imported from the existing game setup file with an ".obj" file extenstion.
Related Use Case(s)	*Scenarios for Import Game Setup Use Case* (Chapter 3.3.1)
Preconditions/Assumptions	Two files exist in the local working directory. One game setup file with file name: "Test.obj" and the other non game setup file with file name: "Test. Txt". Game setup starts and the first game setup window was prompted.
Verification Method	Manual test
User Action(Inputs)	A. 1. Click the "Load Setup File" button. 2. Select the directory, which has files "Test.obj" and "Test.txt". 3. Select file "Test.obj" and Click "Open". 4. Click "Start Game". B. 1. Click the "Load Setup File" button. 2. Select the directory, which has files "Test.obj" and "Test.txt". 3. Select file "Test.txt" and click "Open". 4. Click "Start Game". C. 1. Click the "Load Setup File" button. 2. Select the directory, which has files "Test.obj" and "Test.txt". 3. Select file "Test.obj" and Click "Open". 4. Click "Cancel".
Expected Results	A. Game setup file "Test.obj" is loaded successfully. B. Non game setup file "Test.txt" is not loaded successfully. C. Importing game setup file "Test.obj" is cancelled.
Trace Log Information	None.
Actual Results	Same as the expected result.
Post condition	A. The setup values outlook of Test.obj is displayed in the IT department outlook window. Game play starts with all these setup values. B. Error message" The file is not the proper object file" is shown. C. The setup values outlook of Test.obj is displayed in the IT department outlook window. System goes back the game setup window and game play didn't start.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Based on the use cases scenarios, which are described in the chapter 3.3.1, the above testing procedure takes three possible series of user actions into account. Accordingly, three possible expected/actual results and pre/post conditions are described. The above test case is passed, since the actual result and the expected result is same. There is no trace log information, because the program is not coded to display extra information in the back-end DOS command window.

7.4.2 Game Play Test Cases

For the game play tests, we present two most important test cases. The detail is shown in the table 7.4 and 7.5 respectively. The testing procedure in table 7.4 describes a manual test for making department decisions. In order to test if decisions can be made and the result of the decisions can be obtained right away, the following testing procedure are made based on the user cases described in section 3.3.2.

Table 7.4 Testing Procedure for Making Department Decision

Test Procedure ID	P01
Test Name	Make department decisions
Test Procedure Author	Maya & Kevin
Test Objective	To test if we can make decisions for the department properties and if the result of the decision can be displayed right away. The current changeable department properties are the education days and the retention budget.
Related Use Case(s)	*Scenarios for Make Department Decisions Use Case* (Chapter 3.3.2)
Preconditions/Assumptions	1. "Test.obj" was loaded to game play. 2. IT department game play window was prompted. 3. IT department details and profession list were displayed in this window.
Verification Method	Manual test
User Action(Inputs)	A. 1. Select education days input text field and input 20 for it. 2. Select retention budget input text field and input 2000 for it. 3. Click "Submit". B. 1. Select education days input text field and input 20 for it. 2. Select retention budget input text field and input 2000 for it. 3. Click "Submit". 4. Click "Reset".
Expected Results	A. Education days are changed to 20 and retention cost are changed to 2000. B. Before resetting, education days are changed to 20 and retention cost are changed to 2000. After resetting, education days are recovered to 10 and retention cost is recovered to 1000.
Trace Log Information	None.
Actual Results	Same as the expected result.
Post condition	A. The value of education days is displayed with 20 and the value of retention cost is displayed with 2000. Department decision can be made and submitted again. B. The value of education days is displayed with 10 and the value of retention cost is displayed with 1000. Department decision can be made and submitted again.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Based on the use cases scenarios, which are described in section 3.3.2, user actions can be to submit decisions or to reset the modified properties. Once the user submits the decisions or resets the modified properties, those properties will be changed right away and displayed in the department.

The testing procedure in Table 7.5 describes a manual test for making profession decisions. The decisions we made should be saved as a temporary decisions. The decisions will not be submitted until we finish the current game period. Therefore, those temporary decisions can be modified or cancelled until we finish the current game period. The decisions we made only take place just after the current game period finishes. The result of decisions is displayed in the next game period by recalculating the profession details. Therefore the next game period has the outlook about the previous profession details, the decisions we made and the current profession details.

Table 7.5 Testing Procedure for Making Profession Decision

Test Procedure ID	P02
Test Name	Make profession decision
Test Procedure Author	Maya & Kevin
Test Objective	To test if we can make decisions for each selected profession and if these decisions will take place just after this game period. We are going to test basic decisions like, fire/fire, subcontract/terminate and Outsource/Terminate.
Related Use Case(s)	*Scenarios for Make Profession Decisions Use Case* (Chapter 3.3.2)
Preconditions/Assumptions	1. "Test.obj" was loaded to game play. 2. One profession was selected from the profession list and "details" button was clicked. Here is profession "Development Manager". 3. Current profession details of profession "Development Manager" were displayed in the profession "Development Manager" window. Basically there are 0 FTR, 0 obsolete FTR, 1 FTS and 0 FTO.
Verification Method	Manual test
User Action(Inputs)	A. 1. Hire 3 new FTR employees. (fill 3 in the text field "hire new" FTR) 2. Terminate 1 people for FTS. (fill 1 in the text filed "terminate current contract for") 3. Make new contract with 1 people for FTO. (fill 1 in the text field "make new contract with") 4. Click "Ok". 5. Click"details" in the IT department Game play window. 6. Click "Ok". 7. Click "Finish" in the IT department Game play window. 8. Select "Development Manager". 9. Click "View Detail". B. 1. Hire 3 new FTR employees. (fill 3 in the text field "hire new" FTR) 2. Terminate 1 people for FTS. (fill 1 in the text filed "terminate current contract for") 3. Make new contract with 1 people for FTO. (fill 1 in the text field "make new contract with") 4. Click "Cancel". 5. Click" details" in the IT department Game play window. 6. Click "Ok". 7. Click "Finish" in the IT department Game play window.

	8. Select "Development Manager". 9. Click "View Detail".
Expected Results	A. Hire 3 new FTR, Terminate 1 FTS and outsource 1 FTO are set as a temporary decision in the current game period. After entering the next game period, The decisions are carried out, and the recalculated profession details are displayed. In this case, we will have 2 FTR, 1 obsolete FTR, 0 FTS and 1 FTO; because one of the 3 fired FTR becomes an obsolete people based on the business rules. B. No temporary decisions are made and the system keeps the default value in the current game period. Since no decisions are made, The profession details recalculation will be done based on the default values. In our case the new profession detail will be 0 FTR, 0 obsolete FTR, 1 FTS and 0 FTO.
Trace Log Information	None.
Actual Results	Same as the expected result.
Post condition	A. Game is in the next period. Profession details are displayed with 2 FTR, 1 obsolete FTR, 0 FTS and 1 FTO. It's ready to make decision for the current game period. B. Game is in the next period. Profession details are display with 0 FTR, 0 obsolete FTR, 1 FTS and 0 FTO. It's ready to make decision for the current game period.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Chapter 8

CONCLUSIONS AND FUTURE WORKS

8.1 Introduction

This chapter is going to present the conclusions of this project and suggest the possible future works. Section 8.2 discusses the result of this project and makes conclusions. Section 8.3 gives the recommendations of future works.

8.2 Conclusions

The main objective of this project is defined as following,

To create SHAPE Workforce Management Game (SHAPE WMG) that can simulate the effects of the player's decision based on the current condition of the department by applying Synthesis-Based Software Architecture Design (Synbad) method.

By going through the architecture design and implementation phase, this objective has been accomplished.

8.2.1 Conclusions on Architecture Design

As described in section 1.3, there are several questions addressed on the assessment of the architecture design:

- How well does the architecture solve all the problems arise in the game playing requirements?
- How stable is the architecture for changes that might happen in the future?
- How is the reusability aspect of the architecture?

From the testing that is done with the user, it has shown that the application that are implemented based on the designed architecture is able to fulfill the game playing requirements that are described by the user in the beginning of the project. The game is able to provide the business environment that resembles the real business situation where the game player works in, by allowing IBM consultants to set the game world in the game setup phase. The game is also able to perform the tasks of simulating the results of the game player's decisions by applying the business rules to the current situation of the department where the game player works in.

The evaluation that was done on the architecture design proved that the architecture is indeed stable and reusable. The changes that are required in a certain component to perform certain evolution scenario do not result in big changes that should also be made in other components. The existing components can also be used to add certain new functions or operations to give new features in the game.

8.2.2 Conclusions on Design Methodology

As also described in section 1.3, the assessment of the design methodology is aiming at the ease of use of the Synbad method.

While the user requirements analysis is used to understand and represent directly the problem of concern, the technical problem analysis is used to identify the real problems that will arise in the implementation of the application. The mapping from user requirements into technical problems that was done in Synbad method has helped in identifying the core of the problems that were then solved in the form of software architecture.

In searching for the solution domains for every sub-problem that was identified in the technical problem analysis, several difficulties were found. Due to the limited time and experience that were available to do both the design and implementation of the application, not all solution domain concepts could be refined. However, the architecture design has been proved to be able to cover the problems that were identified before. The clear definition of the solution domain concepts has helped to give novel design solutions. These concepts can be further refined in the future to add more functionality to the application.

8.3 Recommendations for Future Works

From the above conclusion, both architecture design and design methodology can be improved and enhanced in the future. Section 8.3.1 gives recommendations for future works on the architecture design and section 8.3.2 gives recommendations for Synbad as the design methodology that is used in this project.

8.3.1 Future Works on Architecture Design

The *Business Modelling* and *Simulation Modelling* part of the overall SHAPE WMG architecture have been developed with the internal structure analysis. The *Game Player Modelling* part which is also shown in Figure 5.8, however, is not yet developed in the current stage. Further works should be done to design the internal architectures of this part in the lower abstraction level.

In the future, a more complete and realistic business model should be designed. Therefore, the solution domain analysis of the business model needs to be done in more detail. On one side, new elements can be added into the business model, and on the other side, the current primary elements can be enhanced and updated.

As mentioned before, the use of design pattern has helped in solving specific design problems and has made object-oriented designs more reusable. However, since the principles of design patterns are new for us to learn, there was not enough time to understand the concept, to find the pattern that fits properly into the design of this project and to experience the design itself in the

implementation. The only design pattern that is applied in the architecture design is the *Observer* pattern. For future development, some more architecture implementation patterns should also be explored to see whether or not it can be applied in SHAPE WMG application in order to give much higher level of reusability.

8.3.2 Recommendations for Synbad

Based on the experience in applying Synbad in this project, several recommendations are made:

- In the first process of doing the Synbad approach, there was issue discussed on whether to do the technical problem and solution domain analysis with breadth-first approach or depth-first approach. In breadth-first approach, all the problems are worked out in the higher level before continuing with the lower level of defining problems and solution domains. The depth-first approach suggests first to solve the problems with higher priority from the higher to the lower level and then go to the next lower priority problems and solves it in the same manner. Since this project was done by two people, every step of Synbad is always divided into two tasks that can be assigned each one person. This includes the technical problem and solution domain analysis step where parallel work was made to identify problems each in a specific area and find the solution domains for each area. With this breadth-first approach, we found some difficulties in having to redesign the solutions whenever the separate solution domains for each area are combined and didn't fit with each other. Therefore, we suggest that first-depth search is used to save more time in solving the lower priority problems, since the higher ones are already solved and can be used as the basis to solve the other problems.
- Another difficulty that we found in applying Synbad method is in trying to refine all the concepts that were described when solving the problems in the solution domain analysis phase. Since the project demands a deliverable implementation that is ready to use within a limited time, we finally left out several concepts to be able to fulfil this demand but still able to support high reusability of the design. Therefore, we think that Synbad method should provide a flexible way to determine which part of the architecture should be developed first and which part could be developed later on in order to save time in developing the application to fulfil first the main functionality of the system.
- The searching of solution domain has taken quite an amount of time. Finding the right solution domain concepts to be applied in a certain situation is very critical and needs a lot of experience and knowledge. When the project is in the lack of time, some experts help in this area should be used to make the process more effective and at the same time add more experience and knowledge to the designers themselves.

Appendix A

SCENARIOS FOR USE CASES IN THE GAME SETUP PHASE

In this appendix, the scenarios are explained for every use cases that are exist in the game setup phase. The description of scenarios is the result of the requirement analysis phase of the Synbad method as described in Chapter 3.

Scenarios for Validate User Use Case

Scenario 1: User is validated successfully

1. User chooses start game set-up action.
2. System asks for user name and password.
3. User enters user name and password.
4. System looks up the user name and password in its password log.
5. The user name and password match with the entries in the log, the systems then validates the user as the game manager.

Scenario 2: Validation failed since the user is not an authorized game manager.

1. User chooses start game set-up action.
2. System asks for user name and password.
3. User enters user name and password.
4. System looks up the user name and password in its password log.
5. The user name and password don't match with any entries in the log, the systems then informs the user that validation fails.
6. System rolls the user back to the previous state before choosing to start game set-up action.

Scenarios for Look Up Help Use Case

Scenario 1: System displays the selected help content

1. Game manager chooses the "read help" action.
2. Game manager chooses to read help content.
3. Game manager chooses one of the help topics.
4. System displays the help content of the chosen topic.

Scenario 2: System shows the topics related to the keyword entered by the game manager

1. Game manager chooses the "read help" action.
2. Game manager chooses to search for topic.
3. Game manager enters a keyword.

4. System looks for topics that are related to the keyword.
5. System displays the topics related to the keyword entered.

Scenario 3: System cannot find any topic related to the keyword entered by the game manager

1. Game manager chooses the "read help" action.
2. Game manager chooses to search for topic.
3. Game manager enters a keyword.
4. System looks for topics related to the keyword
5. There are no topics related to the keyword, system then informs the user that there are no topic found for the entered keyword.

Scenario 4: System displays the selected tutorial

1. Game manager chooses the "read help" action.
2. Game manager chooses to read tutorial.
3. Game manager chooses one of the tutorials.
4. System displays the chosen tutorial.

Scenarios for Import Game Setup Use Case

Scenario 1: System sets up all the values based on the imported game setup file

6 Game manager chooses the "import game setup" action.

7 Game manager selects a file.

8 System checks to see if the imported file is a valid game setup file.

9 The imported file is a valid file, system then loads the game setup file.

10 The setup values are set to the values that are in the game setup file.

Scenario 2: System fails to set the values since the imported file is not a valid game setup file

11 Game manager chooses the "import game setup" action.

12 Game manager selects a file.

13 System checks to see if the imported file is a valid game setup file.

14 The imported file is not a valid file.

15 System informs the game manager that the file cannot be loaded.

Scenario 3: There are no values set since the game manager cancels the import file action

4. Game manager chooses the "import game setup" action.
5. Game manager cancels the action.
6. System rolls the game manager back to the previous state before the game manager chooses to import game setup.

Scenarios for Manual Game Setup Use Case

Scenario 1: System saves the game setup based on the game manager input in a game setup file

1. Game manager chooses the "setup game manually" action.
2. System displays forms to be filled in by the game manager.
3. Game manager fills in all the required information.
4. Game manager finishes the game setup and sets or chooses a file name to save the game setup.
5. System saves the game setup values in the game setup file.

Scenario 2: There are no values set since the game manager cancels the game setup

1. Game manager chooses the "setup game manually" action.
2. System displays forms to be filled in by the game manager.
3. Game manager fills in all the required information.
4. Game manager cancels the game setup.
5. System rolls the game manager back to the previous state before the game manager chooses to setup the game manually.

Scenarios for Setup Game Period Use Case

Scenario 1: System sets the length and the number of game period based on game manager's input

1. Game manager chooses the length and the number of game period.
2. System sets the chosen values as the length and the number of game period.

Scenario 2: System sets the length and number of game period based on default value

1. Game manager doesn't change the default value of the length and the number of game period that are displayed by the system.
2. System sets the default values as the length and the number of game period.

Scenarios for Setup Department Details Use Case

Scenario 1: System sets the department details values based on game manager's input

1. Game manager gives inputs for all the required values of the department.
2. System sets the department details values as entered by the game manager.

Scenario 2: System sets the department details values based on default value

1. Game manager doesn't change the default value of the department values that are displayed by the system.
2. System sets the default values as the department details values of the game.

Scenarios for Setup Business Strategies Use Case

Scenario 1: System sets the business strategies entered by the game manager

Normal course:

1. Game manager chooses the "setup the game business strategies" action.
2. Game manager types the business strategies for all game periods.
3. System sets the business strategies description as entered by the game manager.

Alternative course:

1. Game manager chooses the "setup the game business strategies" action.
2. Game manager types the business strategies only for some game periods.
3. System informs the game manager that business strategies should be entered for all game periods.
4. Game manager fills in the business strategies description for all game periods.
5. System sets the business strategies description as entered by the game manager.

Scenario 2: System fails to set the business strategies since the game manager cancels the business strategies setup action

1. Game manager chooses the "setup the game business strategies" action.
2. Game manager types the business strategies for some or none game periods.
3. Game manager cancels the business strategies setup.
4. System rolls the game manager back to the previous state before the game manager chooses to setup the business strategies.

Scenarios for Setup Disasters Use Case

Scenario 1: System sets disasters parameter and value based on game manager's input

1. Game manager chooses the "setup disasters" action.
2. Game manager chooses a disaster parameter and sets its value for a game period.
3. System sets the disasters parameter and value for that game period as entered by the game manager.

Scenario 2: System sets the disasters parameters and values based on default value

1. Game manager chooses the "setup disasters" action.
2. Game manager doesn't change the default setting for disaster for a certain game period that is displayed by the system.
3. System sets the default value as the disaster setting for that particular game period.

Scenarios for Add New Profession Use Case

Scenario 1: System adds a new profession set by game manager to the list of available professions

Normal course:

1. Game manager chooses the "setup profession" action.
2. System displays the available professions that can be included in the game.
3. Game manager adds the name of a new profession to the available professions list.
4. System adds the profession name to the available professions list.

Alternate course:

1. Game manager chooses the "setup profession" action.
2. System displays the available professions that can be included in the game.
3. Game manager adds the name of a new profession to the available professions list.
4. The name entered already exists in the list, system then informs the game manager that the profession entered already exists in the list.
5. Game manager enters another profession name to be added.
6. The name entered doesn't yet exist in the list, system then adds the profession name to the available professions list.

Scenario 2: System fails to add new profession since the profession cancels the add new profession action

1. Game manager chooses the "setup profession" action.
2. System displays the available professions that can be included in the game.
3. Game manager adds the name of a new profession to the available professions list.
4. The name entered already exists in the list, system then informs the game manager that the profession entered already exists in the list.
5. Game manager cancels adding new profession to the list.
6. System rolls the game manager back to the previous state before the game manager chooses to add new profession.

Scenario for Remove Profession Use Case

1. Game manager chooses the "setup profession" action.
2. System displays the available professions that can be included in the game.
3. Game manager selects a profession from the list.
4. Game manager chooses the "remove profession" action.
5. System removes the selected profession from the available professions list.

Scenarios for Include Profession Use Case

Scenario 1: System adds a new profession to the list of professions that are included in the game

1. Game manager chooses the "setup profession" action.
2. System displays the available professions that can be included in the game.
3. Game manager selects a profession from the available professions list.
4. Game manager chooses the "include profession" action.
5. System adds the selected profession to the list of professions that are included in the game.

Scenario 2: System fails to add new profession since the selected profession is already included in the game

1. Game manager chooses the "setup profession" action.
2. System displays the available professions that can be included in the game.
3. Game manager selects a profession from the available professions list.
4. Game manager chooses the "include profession" action.
5. The profession selected is already in the list of professions that are included in the game, system then informs the game manager that the selected profession is already included.
6. Game manager adds the name of a new profession to the available professions list.
7. System rolls the game manager back to the previous state before the game manager chooses to add new profession.

Scenario for Exclude Profession Use Case

1. Game manager chooses the "setup profession" action.
2. System displays the list of professions that are included in the game.
3. Game manager selects a profession from the list.
4. Game manager chooses the "exclude profession" action.
5. System removes the selected profession from the list of professions that are included in the game.

Scenarios for Setup Profession Details Use Case

Scenario 1: System sets the profession details values based on game manager's input

1. Game manager chooses the "setup profession" action.
2. System displays the list of professions that are included in the game.
3. Game manager selects a profession from the list.
4. Game manager chooses the "add profession details" action.
5. Game manager gives inputs for all the required values of the profession.
6. System sets the profession details values as entered by the game manager.

Scenario 2: System sets the department details values based on default value

1. Game manager chooses the "setup profession" action.
2. System displays the list of professions that are included in the game.
3. Game manager selects a profession from the list.
4. Game manager chooses the "add profession details" action.
5. Game manager doesn't change the default value of the profession values that are displayed by the system.
6. System sets the default values as the profession details values of the game.

Scenarios for Setup Objectives Use Case

Scenario 1: System sets the objectives values based on game manager's input

1. Game manager chooses the "add profession details" action.
2. Game manager chooses the "setup objectives" action.
4. Game manager inputs the objectives of the profession for all game periods.
5. System sets the objectives values as entered by the game manager.

Scenario 2: System sets the objectives values based on default value

1. Game manager chooses the "add profession details" action.
2. Game manager chooses the "setup objectives" action.
3. Game manager doesn't change the default values of the objectives that are displayed by the system.
4. System sets the default values as the objectives value.

Appendix B

KNOWLEDGE SOURCES FOR SOLUTION DOMAINS

Table B.1 Knowledge Sources for the Solution Domain Control System

ID	Knowledge Source	Form
KS1	*A QoS-Control Architecture for Object Middleware [Ber00]*	Paper
KS2	Meeting with supervisors of UT	Person
KS3	*IBM SHAPE Workforce Management Game External Design [Zan02]*	Document
KS4	Meeting with supervisors of IBM	Person

Table B.2 Knowledge Sources for the Solution Domain Quality Management

ID	Knowledge Source	Form
KS1	Meeting with supervisors of UT	Meeting
KS2	*SHAPE Workforce Management Game [Jol02]*	Thesis report
KS3	*IBM SHAPE Workforce Management Game External Design [Zan02]*	Document
KS4	*Design Patterns: Elements of Reusable Object-Oriented Software [Gam95]*	Textbook

Table B.3 Knowledge Sources for the Solution Domain Object-Oriented Design

ID	Knowledge Source	Form
KS1	*SHAPE Workforce Management Game [Jol02]*	Thesis report
KS2	*WebSphere Version 4 Application Development Handbook [Wah01]*	Technical handbook
KS3	*Java: How to Program [Dei99]*	Textbook
KS4	*An Introduction to Programming and Object-Oriented Design Using Java [Nin02]*	Textbook
KS5	*Design Patterns: Elements of Reusable Object-Oriented Software [Gam95]*	Textbook
KS6	*Applied Java Patterns [Ste02]*	Textbook

Table B.4 Knowledge Sources for the Solution Domain User Interface

ID	Knowledge Source	Form
KS1	IBM SHAPE WMG prototype	Prototype
KS2	*IBM SHAPE Workforce Management Game External Design [Zan02]*	Document
KS3	*SHAPE Workforce Management Game [Jol02]*	Thesis report
KS4	*Java: How to Program [Dei99]*	Textbook
KS5	*An Introduction to Programming and Object-Oriented Design Using Java [Nin02]*	Textbook
KS6	*Design Patterns: Elements of Reusable Object-Oriented Software [Gam95]*	Textbook
KS7	*Applied Java Patterns [Ste02]*	Textbook

Table B.5 Knowledge Sources for the Solution Domain Calculation

ID	Knowledge Source	Form
KS1	*IBM SHAPE Workforce Management Game External Design [Zan02]*	Document
KS2	*SHAPE Workforce Management Game [Jol02]*	Thesis report
KS3	Meeting with supervisors of IBM	Meeting

This knowledge sources for solution domains *Business modeling, Game playing modeling,* and *Assessment and Simulation technique* are the same and are given in the following Table B.6

Table B.6 Knowledge Sources for the Solution Domain Business Modeling, Game Playing Modeling, and Assessment and Simulation Technique

ID	Knowledge Source	Form
KS1	*IBM SHAPE Workforce Management Game External Design [Zan02]*	Document
KS2	*SHAPE Workforce Management Game [Jol02]*	Thesis report
KS3	Meeting with supervisors of IBM	Person
KS4	Meeting with supervisors of UT	Person

Appendix C

TEST CASES FOR THE TESTING PROCEDURE

This appendix describes the test cases used for the testing procedure explained in section 7.4 of Chapter 7.

Table C.1 Testing Procedure for Game Period Setup

Test Procedure ID	S02
Test Name	Game period setup
Test Procedure Author	Maya & Kevin
Test Objective	To test if game period setup can be successfully done.
Related Use Case(s)	*Scenarios for Game Period Setup Use Case* (Appendix A)
Preconditions/Assumptions	"manual setup" has been clicked. Game period setup window was prompted.
Verification Method	Manual test
User Action(Inputs)	A. 1. Choose 6 from the choice button1 for the length of a game period. 2. Choose 2 from the choice button2 for the number of game periods. 3. Click "Next". B. 1. Leave the default values for both choice buttons. 2. Click "Next". C. 1. Choose 6 from the choice button1 for the length of a game period. 2. Choose 2 from the choice button2 for the number of game periods. 3. Click "Back".
Expected Results	A. The length of a game period is set to 6 months. The number of game periods is set to 2. B. The default values are used. The length of a game period is 12 and the number of game periods is 3. C. Game period setup is not done yet. System goes back to the first game setup window.
Trace Log Information	A. The length of a game period is set to 6. The number of game periods is set to 2. B. The length of a game period is set to 12. The number of game periods is set to 3. C. None
Actual Results	Same as the expected result.
Post condition	A. A warning message pops out: "Changing this setting will delete all setup values. Continue?" B. A warning message pops out: "Changing this setting will delete all setup values. Continue?" C. Game Setup window is prompt.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Table C.2 Testing Procedure for Business Strategy Setup

Test Procedure ID	S03
Test Name	Business strategy setup
Test Procedure Author	Maya & Kevin
Test Objective	To test if business strategy setup can be successfully done.
Related Use Case(s)	*Scenarios for Business Strategy Setup Use Case* (Appendix A)
Preconditions/Assumptions	The number of game periods is 3. IT department setup window was prompted. "Business Strategy Setup" was clicked.
Verification Method	Manual test
User Action(Inputs)	A. 1. Input 3 business strategies for 3 game periods respectively. 2. Click "Ok". B. 1. Input 2 business strategies for the first two game periods respectively. 2. Click "Ok". C. 1. Input 3 business strategies for 3 game periods respectively. 2. Click "Cancel".
Expected Results	A. Business strategies are set. B. Business strategies setup cannot be completed because one of game period has not game strategy yet. C. No business strategies are set.
Trace Log Information	None
Actual Results	Same as the expected result.
Post condition	A. Business strategies setup window is closed and system goes back to IT department setup window. B. A warning message pops out, "Please fill in all the required fields". C. Business strategies setup window is closed and system goes back to IT department setup window.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Table C.3 Testing Procedure for Disaster Setup

Test Procedure ID	S04
Test Name	Disaster setup
Test Procedure Author	Maya & Kevin
Test Objective	To test if disaster setup can be successfully done.
Related Use Case(s)	*Scenarios for Disaster Setup Use Case* (Appendix A)
Preconditions/Assumptions	The number of game periods is 3. IT department setup window was prompted. "Disaster Setup" was clicked.
Verification Method	Manual test
User Action(Inputs)	A. 1. Select "Attrition Rate Change" and value is "2%" for the first game period. 2. Select "Illness Rate Change" and value is "3%" for the second game period. 3. Select "None" and value is "2%" for the third game period. 4. Click "Ok". B. 1. Leave all the default values. 2. Click "Cancel".

Expected Results	A. In the first game period, attrition rate will be changed to 2%. In the second period, the illness rate will be changed to 3%. There is no any disaster for the third game period. B. No any changes to the default disaster scenarios. The current default disaster scenarios are all "None" with "Null" values.
Trace Log Information	None
Actual Results	Same as the expected result.
Post condition	A. Disaster setup window is closed and system goes back to IT department setup window. B. Same as A.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Table C.4 Testing Procedure for Adding New Profession

Test Procedure ID	S05
Test Name	Add new profession
Test Procedure Author	Maya & Kevin
Test Objective	To test if a custom profession can be added into the profession list.
Related Use Case(s)	*Scenarios Add New Profession Use Case* (Appendix A)
Preconditions/Assumptions	Profession setup window was prompted. Default profession list was loaded.
Verification Method	Manual test
User Action(Inputs)	A. 1. Click "Add". 2. Fill in a name for the new profession. In this case we fill in "Custom profession". 3. Click "Ok". B. 1. Click "Add". 2. Fill in a name for the new profession. In this case we fill in "CEO", which already exists in the profession list. 3. Click "Ok". C. 1. Click "Add". 2. Fill in a name for the new profession. In this case we fill in "CEO", which already exists in the profession list. 3. Click "Ok". 4. Click "Ok". 5. Click "Cancel".
Expected Results	A. New profession "Custom profession" has been added into profession list. B. Since the new profession name already exists in the profession list. System gives a warning message and ignores the "add" action. C. No new profession has been added, "Add" action is cancelled.
Trace Log Information	None
Actual Results	Same as the expected result.
Post condition	A. System goes back to the profession setup window. Profession list increases one and "Custom profession" appears in the end of profession list. B. A Warning message pops out "The profession name you filled exists already.

	Please choose other profession name to fill in". C. System goes back to the profession setup window. Nothing has been changed.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Table C.5 Testing Procedure for Removing a Profession

Test Procedure ID	S06
Test Name	Remove an existing profession
Test Procedure Author	Maya & Kevin
Test Objective	To test if a custom profession can be removed from the profession list.
Related Use Case(s)	*Scenarios Remove Profession Use Case* (Appendix A)
Preconditions/Assumptions	Profession setup window was prompted. Default profession list was loaded and at least one profession in the list.
Verification Method	Manual test
User Action(Inputs)	A. 1. Select a profession. It is "Operator" in this case. 2. Click "Remove".
Expected Results	A. Profession "Operator" is removed from the profession list.
Trace Log Information	None
Actual Results	Same as the expected result.
Post condition	A. Profession list decreased one and "Operator" has been removed.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Table C.6 Testing Procedure for Profession Detail Setup

Test Procedure ID	S07
Test Name	Profession details setup
Test Procedure Author	Maya & Kevin
Test Objective	To test if profession details can be setup successfully.
Related Use Case(s)	*Scenarios Setup Profession Details Use Case* (Appendix A)
Preconditions/Assumptions	Profession Setup window was prompted. At least two professions are included in the profession list.
Verification Method	Manual test
User Action(Inputs)	A. 1. Select "CEO" from the included profession list. 2. Click "Add Details". 3. Set profession type "Fixed". 4. Fill in 1 for the number of productive FTR. 5. Click "Ok". B. 1. Select "CEO" from the included profession list. 2. Click "Add Details". 3. Set profession type "Changeable and Uncontractable". 4. Fill in 1 for the number of subcontractor. 5. Click "Count Productivity". 6. Click "Ok". C. 1. Select "IT Manager" from the included profession list. 2. Click "Add Details". 3. Set profession type "Changeable and Contractable".

	4. Fill in 1 for the number of productive FTR, 4% for attrition rate, 1 for the number of outsourcer. 5. Click "Count Productivity". 6. Click "Ok". D. 1. Select "CEO" from the included profession list. 2. Click "Add Details". 3. Set profession type "Fixed". 4. Fill in 1 for the number of productive FTR. 5. Click "Cancel".
Expected Results	A. Profession "CEO" has one productive FTR and the profession type is "fixed". B. Profession "CEO" has one subcontractor and the profession type is "Changeable and Uncontractable". The counted productivity is 0.9. C. Profession "IT Manager" has one outsourcer, 4% attrition rate and the profession type is "Changeable and Contractable". The counted productivity is 1.0. D. No properties of profession "CEO" are changed yet, because the actions are cancelled.
Trace Log Information	None
Actual Results	Same as the expected result.
Post condition	A. System goes back to the profession setup window. B. Same as A. C. Same as A. D. Same as A.
Pass/Fail	Pass
Test Environment	English Windows XP, IBM Notebook with P3 CPU, 256M Ram

Appendix D

CALCULATION

As described in subsection 3.2.2, there are three types of profession that are defined in SHAPE WMG:

1. *Fixed* profession: a profession whose number of employees cannot be changed during the entire game and who can only consist of Full Time Regular employees. This profession will be referred to as type 1 profession.
2. *Changeable-uncontractable* profession: a profession whose number of employees can be changed during the game and who can only consist of Full Time Regular employees. This profession will be referred to as type 2 profession
3. *Changeable-contractable* profession: a profession whose number of employees can be changed during the game and who can consist of Full Time Regular employees, Full Time Subcontractors, and Full Time Outsourcers. This profession will be referred to as type 3 profession.

This section will explain the order of calculation that is done in SHAPE WMG. For every calculation step, it is explained to which type of profession it is applied to. The order of calculation in simulating the effects of the decision is as follows:

D.1 Number of Employees Based on Player's Decisions

This step is not applied for profession of type 1.

Full Time Regular

The steps:

1. If the player decided to re-educate obsolete FTR, then the number of the current obsolete is reduced and the number of productive FTR is increased by the number of re-educated obsolete.
2. If the player decided to fire FTR, then the number of remaining obsolete FTR will first be reduced if it is less or equal to the number of fired FTR. If there is still remained a number of fired FTR, then the number of productive FTR will be reduced by that remaining number.
3. If the player decided to hire FTR, then the number of productive FTR is increased with the number of hired FTR.

The algorithm:

if the player decided to re-educate obsolete FTR

new obsolete FTR = previous obsolete FTR – reeducated obsolete

new productive FTR = previous productive FTR + reeducated obsolete

else

new obsolete FTR = previous obsolete FTR

new productive FTR = previous productive FTR

if the player decided to fire FTR

if the number of fired FTR <= new obsolete FTR

new obsolete FTR = new obsolete FTR – fired FTR

else

new productive FTR = new productive FTR – (fired FTR – new obsolete FTR)

new obsolete FTR = 0

new productive FTR = new productive FTR +hired FTR

Full Time Subcontractor

This step is only applied to profession of type 3.

The steps:

1. If the player decided to make new FTS contract, then the number of FTS will be increased by the number of new FTS contract.
2. If the player decided to terminate FTS contract, then the number of FTS will be reduced by the number of terminated FTS contract.

The algorithm:

new FTS = previous FTS +new FTS contract – terminated FTS contract

Full Time Outsourcer

This step is only applied to profession of type 3.

The steps:

1. If the player decided to make new FTO contract, then the number of FTO will be increased by the number of new FTO contract.
2. If the player decided to terminate FTO contract, then the number of FTO will be reduced by the number of terminated FTO contract.

The algorithm:

new FTO = previous FTO +new FTO contract – terminated FTO contract

D.2 Cost

This step is applied to all types of profession.

The steps:

1. Calculate the FTR costs
2. Calculate the retention costs based on the new retention cost that is altered by the player or the initial retention cost if the player didn't alter the cost
3. Calculate the education costs
4. If the profession type is not of type 1 and the player decided to hire FTR, calculate the recruitment costs
5. If the profession type is not of type 1 and the player decided to fire FTR, calculate the golden handshake costs
6. If the profession type is 3, calculate the FTS and FTO costs

The algorithm:

if profession type is 1

FTR costs = productive FTR * compensation FTR

retention costs = productive FTR * new retention cost

education costs = productive FTR * education cost

else

FTR costs = (new productive FTR + new obsolete FTR) * compensation FTR

retention costs = (new productive FTR + new obsolete FTR) * new retention cost

education costs = (new productive FTR + new obsolete FTR) * new education cost

if the player decided to hire FTR

recruitment costs = hired FTR * recruitment cost

if the player decided to fire FTR

golden handshake costs = fired FTR * (compensation FTR +
(golden handshake rate * compensation FTR))

if profession type is 3

FTS costs = new FTS * compensation FTS

FTO costs = new FTO * compensation FTO

D.3 Attrition Rate

This step is not applied to profession of type 1.

The steps:

1. If there is disaster scenario set to change the attrition rate, then the number of current attrition rate is increased or decreased by the attrition rate change.

2. If the player decided to alter the number of education days, then the following rules will be applied:
    - if the new number is higher, then the attrition rate will increase 1 % for each 10 % of the deviation between the previous and the new number of education days
    - if the new number is lower, then the attrition rate will decrease 1 % for each 10 % of the deviation between the previous and the new number of education days
3. If the player decided to alter the retention cost, then the following rules will be applied:
    - if the new cost is higher, then the attrition rate will decrease 1 %
    - if the new cost is lower, then the attrition rate will increase 2 %

The algorithm:

Based on Disaster

if there is disaster applied on attrition rate

new attrition rate = previous attrition rate + attrition rate change

else

new attrition rate = previous attrition rate

Based on Education Days Granted

if number of education days is altered

deviation = ((new education days – previous education days) / previous education days) * 100 %

new attrition rate = new attrition rate – (deviation / 10 %)

Based on Retention Cost Spent

if retention cost is altered

if new retention cost > previous retention cost

new attrition rate = new attrition rate – 1 %

else

new attrition rate = new attrition rate + 2 %

D.4 Other Leavers

This step is not applied to profession of type 1.

The steps:

1. Calculate the number of leavers by applying the attrition rate (obtained form the attrition rate calculation in section 2)
2. If the player decided to fire FTR equal to or more than 50% of the total FTR, other FTR that are 25 % of the initial number of FTR (before firing) will also leave.
3. Reduce the number of productive FTR by the number of other leavers.

The algorithm:

other leavers = new attrition rate * (new productive FTR + new obsolete FTR)

if player fired FTR more than 50 %

other leavers = other leavers + 25 % * (new productive FTR + new obsolete FTR)

new productive FTR = new productive FTR – other leavers

D.5 Illness Rate Based on Disaster Scenario

This step is not applied to profession of type 1.

The steps:

1. If there is disaster scenario set to change the illness rate, then the number of current illness rate is increased or decreased by the illness rate change.

The algorithm:

if there is disaster applied on illness rate

new illness rate = previous illness rate + illness rate change

else

new illness rate = previous illness rate

D.6 Productivity

This step is not applied to profession of type 1.

The steps:

1. Calculate the current number of productive FTR by reducing it with the other leavers.
2. Calculate the productivity of FTR with three different formulas: for the newly hired FTR, for the re-educated obsolete and for the remaining FTR:
3. Calculate the total FTR productivity
4. If the profession is of type 3, then calculate the productivity for FTS and FTO with three different formulas: for new FTS, foe the remaining FTS, and for FTO
5. Calculate the total FTS productivity
6. Calculate the total productivity of the profession

The algorithm:

new productive FTR = new productive FTR – other leavers

hired FTR productivity = hired FTR * ((260.0 – new education days – vacation days – learning curve FTR - (2.6 * new illness rate))/260.0)

re-educated FTR productivity = re-educated obsolete * ((260.0 – new education days – vacation days – learning curve obsolete - (2.6 * new illness rate))/260.0)

remaining FTR productivity = (new productive FTR – hired FTR – re-educated obsolete) *
((260.0 – new education days – vacation days –
(2.6 * new illness rate))/260.0)
total FTR productivity = hired FTR productivity+ re-educated obsolete productivity +
remaining FTR productivity
total productivity = total FTR productivity

if profession type is 3

new FTS productivity = new FTS contract * ((260.0 – vacation days –
learning curve FTS)/260.0)
remaining FTS productivity = (new FTS – new FTS contract) * ((260.0 –
vacation days)/260.0
total FTS productivity = new FTS productivity + remaining FTS productivity
total FTO productivity = new FTO
total productivity = total productivity + total FTS productivity +
total FTO productivity

D.7 New Obsolete

This step is not applied to profession of type 1.

The steps:

1. If the accomplished productivity is more than the needed productivity, it will result in new obsolete FTR which is the difference between those productivities
2. If the difference is less than or equal to the number of current productive FTR, then the number of obsolete is increased by that difference and the number of productive FTR is decreased also by that difference. If the difference is more than the number of current productive FTR, then the number of obsolete FTR is increased by the number of productive FTR and the number of productive FTR is set to zero.

The algorithm:

if the accomplished productivity is more than the needed productivity

new obsolete = accomplished productivity – needed productivity
if new obsolete <= new productive FTR
new productive FTR = new productive FTR – new obsolete
else
new obsolete = new productive FTR;
new productive FTR = 0

D.8 Illness Rate Based on Productivity

This step is not applied to profession of type 1.

The steps:

1. Calculate the deviation of the accomplished productivity from the needed productivity.
2. If the deviation is more than 10%, then first check the current game period:
    - If the current period is 1, then the illness rate is increased by 3 %
    - If the current period is 2 or larger, then check if the illness rate has never been increased before or if it has but then decreased again. If so, then the illness rate is increased by 3 %
3. If the deviation is less than or equal to 10%, then first check the current game period:
    - If the current period is less than 3, then do nothing (because illness rate is only decreased after it has been increased before and should be at least 2 periods apart)
    - If the current period is 3 or larger, then check if the illness rate has already been increased before at least two periods before and not decreased again afterwards. If it has then the illness rate is not changed, but if it has not then the illness rate is decreased by 3 %

The algorithm:

deviation = (accomplished productivity – needed productivity) / needed productivity

if the deviation > 10 %

    if the current period is 1

        new illness rate = new illness rate + 3 %

    else

        if the illness rate is not increased in previous periods or it was but then decreased

            new illness rate = new illness rate + 3 %

else

    if the current period is 3 or larger

        if the illness rate is increased at least two periods before and not decreased then

            new illness rate = new illness rate - 3 %

REFERENCES

[Aks02] Akşit, Mehmet and Bedir Tekinerdoğan. *Ontwerpen van Software Architecturen*, Universiteit Twente, July 2002.

[Bei90] Beizer, B., *Software Testing Techniques (second edition)*, Thomson Computer, Press, Boston, ISBN 1850328803, 1990

[Bei95] Beizer, B., *Black box testing: Techniques for Functional Testing of Software and Systems,* John Wiley & Sons inc., New York, ISBN 1-872582-17-6, 1995

[Ber00] Bergmans, L., A. van Halteren, L. Ferreira Pires, M. van Sinderen, and M.Aksit, *A QoS-Control Architecture for Object Middleware,* CTIT, University of Twente, The Netherlands, 2000.

[Boo99] Booch, Grady, James Rumbaugh and Ivar Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[Dei99] Deitel, H.M. and P.J. Deitel. *Java: How to Program*, Prentice Hall, 3rd Edition, 1999.

[Dus03] Dustin, E (2003) Effective Software Testing, 50 specific ways to improve your testing, Addison-Wesley, Pearson Education, Inc, ISBN 0-201-79429-2

[Gam95] Gamma, Erich, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Hut03] Hutcheson, M, L (2003) Software Testing Fundamentals, Wiley Publishing Inc, Canada, ISBN 0-471-43020-X

[Jac99] Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison-Wesley, 1999.

[Jol02] Joling, H.M. *SHAPE Workforce Management Game*, Technische Informatica, Hogeschool Drenthe, 2003.

[Kru01] Kruchten, Philippe. *What is the Rational Unified Process?*, The Rational Edge, 2001. http://www.therationaledge.com/content/jan_01/f_rup_pk.html

[Lev02] Levitt, Karl. *Introduction to Software Engineering: class notes from Tuesday 5/14/02*, 2002. http://wwwcsif.cs.ucdavis.edu/~cs160/ECS160A-notes-10%20copy.pdf.

[Nin02] Niño, Jaime and Frederick A. Hosch. *An Introduction to Programming and Object-Oriented Design Using Java*, John Wiley & Sons, Inc. , 2002.

[Rop94] Roper, M. (1994), *Software Testing,* McGraw-Hill, England, ISBN 0-07-707466-1Simmons, T. (2000), A bug's life, in: *Professional Tester*, Volume 1, and Issue No.4

[Sie96] Siegel, S (1996), *Object Oriented Software Testing, A Hierarchical Approach*, John Wiley & Sons, Inc, Canada, ISBN 0-471-13749-9

[Ste02] Stelting, Stephen A. and Olav Maassen. *Applied Java Patterns*, Pearson Higher Education, 2002.

[Tek00] Tekinerdoğan, Bedir and Mehmet Akşit. *Synthesis-Based Software Architecture Design*, Ph.D. Thesis, Department of Computer Science, University of Twente, 2000.

[V2M2] Verification and Validation Maturity Model, *Test Techniques and Methods*, Faculteit Technologie Management, Technische Universiteit Eindhoven. http://tmitwww.tm.tue.nl/research/v2m2/workarea/wp2/Test%20Techniques%20and%20Methods.pdf.

[Wah01] Wahli, Ueli, Alex Matthews, Paula Coll Lapido, and Jean-Pierre Norguet. *WebSpehere Version 4 Application Development Handbook*, IBM, 1st Edition, September 2001.

[Zan02] Zandvliet, Keimpe. *IBM Shape Workforce Management Game External Design*, IBM, 2002.