A Service Oriented Architecture Solution for Gaming Simulation Suites

Master Thesis Report

Bas van Nuland

A Service Oriented Architecture Solution for Gaming Simulation Suites

THESIS REPORT

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Bas van Nuland
born in Zeven, Germany

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`http://www.ewi.tudelft.nl/`

Faculty of Technology, Policy and Management
Jaffalaan 5
Delft, the Netherlands
`http://www.tbm.tudelft.nl/`

A Service Oriented Architecture Solution for Gaming Simulation Suites

Author: Bas van Nuland
Student id: 1150448
Email: B.vanNuland@student.tudelft.nl

Abstract

Serious Gaming is becoming a popular method for training and problem solving in companies. One of the companies who has taken an interest in this is ProRail. Together with the faculty of Technology, Policy and Management of the Delft University of Technology they started a project to develop a gaming simulation suite for training and decision making purposes, called the Railway Gaming Suite. In order to connect the games and simulators of the suite a solid architecture is needed. Three architectures were picked out to see if they are suitable for this, namely: Service Oriented Architectures, High Level Architecture and FAMAS Simulation Backbone.

Using the Railway Gaming Suite as a case study, we have extracted requirements (like performance and flexibility) for an architecture for gaming simulation suites using the Architectural Trade-off Analysis Method. These requirements are used to determine the suitability of the three architectures. In this thesis the research on the suitability of Service Oriented Architectures (SOA) is presented. A prototype SOA was created, called Service Oriented Gaming and Simulation (SOGS). This prototype was used to test the performance requirement for the evaluation. The suitability was investigated by evaluating SOA to see if it is able to support the requirements we found. We subsequently also compared the suitability of the other architectures. Intermediate results of this thesis project were used to help with the decision for selecting an architecture for the Railway Gaming Suite.

Thesis Committee:

Chair:	Prof. Dr. Arie van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Dr. Andy Zaidman, Faculty EEMCS, TU Delft
Company supervisors:	Prof. Dr. Ir. Alexander Verbraeck, Faculty TPM, TU Delft

Preface

This thesis represents the end result of the work I have done for my master's project in the last nine months. The subject of this thesis was proposed to me by Rens Kortmann when I worked for him on one of the games for the Railway Gaming Suite. My interest in serious gaming and software engineering made this a perfect subject. Parallel to my research into Service Oriented Architectures, research studies were performed for two other architecture (High Level Architecture and FAMAS Simulation Backbone) in the same context. This gave me the opportunity to broaden the scope of my thesis by comparing my work with the work on the other architectures.

This thesis report has been made possible with the support of many people. First, my colleagues at the Game Lab at TPM who kept me sharp during the long days behind my desk. Then the researches of the Systems Engineering and Policy Organization Law and Gaming; Sebastiaan Meijer, Mamadou Seck, Cagri Tekinay and Sibel Ecker. In particular I would like to thank my supervisors; Rens Kortmann and Alexander Verbraeck. Next I want to extend my gratitude to my supervisor of the Software Engineering Research Group, Department of Software Technology at the Faculty EEMCS, Andy Zaidman, who has helped me a great deal during this thesis and especially with writing this report. Furthermore I would like to thank my parents for supporting me through all my years at the TU Delft. Last, the person who I am very grateful to is Suzanne Vaartjes, who supported and pushed me during this thesis and ensured I finished it.

Bas van Nuland
Delft, the Netherlands
27 April, 2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For years games have been seen as a leisure activity. In the 20th century this trend continued and found its way in the computer industry. Towards the end of the nineties, the digital gaming industry has surpassed the movie industry in Hollywood as one of the leading entertainment industries in the world [11]. From Pong[1] to Unreal Tournament[2] games became more and more interactive and realistic. With the growth of the Internet multi-player gaming transformed from two people sitting at the same computer to 'massively multiplayer online games' in which thousands of people play the same game together [28].

In the mean time 'serious' industries are impacted by game and computer game technologies. For example in military, soldiers are trained using virtual environments. The Dutch railway company ProRail also has taken an interest in using computer games to help with training and decision making. This resulted in a collaboration of ProRail with the faculty of Technology, Policy and Management of the Technical University Delft. Part of this collaboration is the development of a gaming simulation suite called the Railway Gaming Suite (RGS).

The Railway Gaming Suite requires the connection of newly developed games with different simulators of the physical infrastructure and the control systems that are already developed and in use by ProRail. To facilitate this connection between different software applications a work package is initiated by a team of researchers at the TU Delft to look at some architectures for distributed simulation and gaming. The research done for this thesis report is part of this project. The main goal for this work package in the ProRail RGS project has been defined as: "To understand what system architectures are suitable for distributed, decision supporting games and simulations in the domain of rail traffic control"[22].

At the start of the work package it was decided to take a look at three architectures; High Level Architecture (HLA), FAMAS Simulation Backbone, and Service Oriented Architectures (SOA). HLA is an architecture specifically designed for simulation gaming by the Department of Defence of the United States. HLA is part of the research because of this background and ProRail has some experience with it in previous projects. FAMAS Simulation Backbone is an architecture developed as part of a PhD thesis [9]. The aim of the FAMAS Simulation Backbone is to provide a flexible architecture for

[1] *http://en.wikipedia.org/wiki/Pong/*
[2] *http://en.wikipedia.org/wiki/Unreal_Tournament/*

the interoperability among various distributed simulation models. This makes it interesting to research in the RGS work package. SOA is a popular architectural paradigm often used by enterprise architects [7]. It has no direct connection to simulation gaming, but the the SOA paradigm supports interoperability and flexibility in a distributed environment. The service oriented paradigm has been around for a couple of years and has become increasingly popular with organizations. SOA has promised to deliver unprecedented flexibility and cost savings to IT, by defining a methodology for the use and re-use of software components and business processes [8]. This makes it an interesting architectural paradigm to research for gaming simulation suites. Research into these architectures has been divided among different members of the Railway Gaming Suite team.

In this thesis report research of the service oriented architecture solution will be the focus. The other two architectures will be researched by other members of the RGS team. We use the Railway Gaming Suite as a case study for this research. By looking at the RGS we try to extract general requirements for gaming simulation suites and see how these are supported by the SOA paradigm.

In the next section we will look at the research questions for this thesis project.

1.1 Research Questions

The goal of the first work package of the Railway Gaming Suite project is finding a suitable architecture for the RGS. One of the candidate architectures is SOA. In this thesis we look at the suitability of SOA for the Railway Gaming Suite and try to generalize this to gaming simulation suites. This leads to the main research question for this thesis project:

- **RQ** Is SOA a suitable architecture for gaming simulation suites?

In order to answer this question we need to know when an architecture is suitable, if the SOA paradigm supports the suitability requirements and how it compares to other architectures. This leads to the following three sub questions:

- **RQ1** What are the architectural requirements to determine the suitability of gaming simulation suites?

- **RQ2** How well does a service oriented architecture support the requirements of gaming simulation suites?

- **RQ3** How does a service oriented architecture compare to other architectures?

To answer the first research question we need background information on gaming simulation suites and architectural analysis methods, this results in the following questions:

- **RQ1.1** What are gaming simulation suites?

- **RQ1.2** What is a good method to determine architectural requirements of a system?

For the second research question we need information on service oriented oriented architectures, and possible implementations to test the requirements. Thus we come to the next questions:

- **RQ2.1** What are the principles of a service oriented architecture?
- **RQ2.2** Can we construct a prototype SOA to test performance requirements?

For the final research question we need to know more about the distributed simulation domain and if there are already architectures for gaming simulation suites and how well they support the requirements:

- **RQ3.1** Are there architectures currently in use for gaming simulation suites?
- **RQ3.2** What are the architectural approaches of these architectures?
- **RQ3.3** How well do the other architectures support the requirements of gaming simulation suites?

1.2 Outline of the Report

We start this thesis report by looking at gaming simulation suites in chapter 2. In particular we will look at the Railway Gaming Suite project. Using the RGS as a case study we determine the architectural requirements in chapter 3. This chapter will explain the method we used to determine the requirements. This method is called Architectural Trade-off Analysis Method (ATAM). After this we take a look at the different architectures that are part of the RGS work package. First we will discuss the SOA paradigm in chapter 4. To better research the suitability of the service oriented architectures a prototype architecture has been designed and implemented. It is called Service Oriented Gaming and Simulation (SOGS) and is the topic of chapter 5. Then we give an overview of HLA and FAMAS in chapter 6. In the same chapter we take a quick look at entertainment gaming. Using the information of these chapters an evaluation of the three different architectures is performed in chapter 7. The results of the evaluation are discussed in chapter 8. Finally chapter 9 gives the conclusion of this report.

Chapter 2

Gaming Simulation Suites

In order to evaluate whether a service oriented architecture is a good distributed architecture for simulation gaming suites we first need to know what a simulation gaming suite is, thus answering research question:

- **RQ1.1** What are gaming simulation suites?

In this chapter we look at the definition of the term *gaming simulation suite* and we present the Railway Gaming Suite as an example of such a gaming simulation suite. The Railway Gaming Suite will be used throughout this report as a case study in order to find out what is important for gaming simulation suites. Using the important aspects of the RGS we set out to define some general requirements for gaming simulation suites.

2.1 Serious Games and Simulation Games

We can divide the term gaming simulation suite in two parts, 'gaming simulation' and 'suite'. We start with the last part 'suite', which is easy to explain. A suite in the software world is a collection of applications. That leaves us with 'gaming simulation' to specify what kind of applications are in the collection. As the name suggests here we are concerned with simulation games, which are serious games with a focus on training and prediction. This is further explained below. So in short a gaming simulation suite is a collection of simulation games. Now that we have defined the type of applications we can take a deeper look into it.

Simulation games are a subcategory of a specific kind of games, called serious games. In his book 'Serious Games', Clark C. Abt[3] explains the idea behind it nicely:

> The oxymoron of Serious Games unites the seriousness of thought and problems that require it with the experimental and emotional freedom of active play. Serious games combine the analytic and questioning concentration of the scientific viewpoint with the intuitive freedom and rewards of imaginative, artistic acts.

The definition he uses is:

> Reduced to its formal essence, a game is an activity among two or more independent decision-makers seeking to achieve their objectives in some limiting context. A more conventional definition would say that a game is a context with rules among adversaries trying to win objectives. We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement.

Even though Clark Abt was talking about serious games as early as the 70's, the term was not actively used until the Serious Games Initiative[1] was formed in 2002. Even though there is no official list of categories of serious games, there are some terms that are widely used, such as simulation games. Examples of other used categories are; military games (used in the military) and persuasive games (games used as persuasion technology). In simulation games the focus lies on training or prediction. So a gaming simulation suite can be defined as a collection of games used for training or prediction.

Throughout this thesis we will use a number of terms related to gaming and simulation. In the context of this report, these terms have as specific meaning. To make sure this meaning is clear a short explanation is given:

- Game - a game is a software application where there is a human controlling at least part of what happens in the application.
- System - the collection of all games and simulators that are used for a scenario is called the system.
- Suite - the collection of all games and simulators that can be use to set up a system.
- Simulator - a simulator is a software application that plays out a scenario according to predefined data. It is not directly influenced by a player. Data can be changed through indirect means.
- Scenario - a scenario is the set-up of the system. The scenario specifies which games and simulators are used and the case that will be played.
- Session - actually playing a scenario is called a session.
- Facilitator - the facilitator guides a session. This can be done by verbal instructions (for example a briefing) of by influencing the system itself.

Other important concepts in simulation gaming are real-time, synchronization and causality. These require a little more explanation and are further described below. There are other techniques that can be used, like packet compression and aggregation, interest management and dead reckoning. These are however not specific to simulation gaming, but more to distributed gaming environments in general. They will be covered in section 6.3 on multi-player entertainment gaming.

[1] *http://www.seriousgames.org/*

2.1.1 Real-time

The term real-time is used in several ways in software engineering. Real-time computing for example is concerned with systems with strict time constraints (e.g. air bags in a car). In gaming real-time is used when the game is continuous. In this thesis we use real-time when the time in the game is continuous and synchronized with wall clock time.

2.1.2 Synchronization

In a gaming simulation environment it is important to be able to synchronize events of different applications. This is important because the applications might need information from other applications before continuing themselves. Without this synchronization an application can do calculations based on incomplete information and thus makes incorrect calculations. This may result in having to redo the calculations or worse, faulty data in the system. There are several ways of synchronizing all applications. We describe three methods to do this below.

One way to synchronize is by keeping a centralized list of all critical messages sent in the system. From this central place the messages can then be send to the receivers. When all critical messages are received by an application it can receive a ’go’ from the central hub to do its next calculations. The central hub is in control of allowing an application to continue. Because it has a complete overview and complete control of all messages sent in the system it can guarantee synchronization. All messages that are sent through the system need to go to the central hub, from there they are sent to the receiver(s). We call this a *central synchronization mechanism*.

A problem arises when the services connected to the applications each have their own message list. Somehow these lists should be synchronized to make sure the applications are up-to-date. A way to do it is to have a central synchronization application which tells all services to perform the calculation they can do at the moment, then if there are messages, send them. The synchronization service has to check if all messages are received and can then tell the application to proceed, go to the next step. Before the application can now start with its calculations they first need to process the messages received in the previous step. From here it starts all over again. This way all calculations are based on the most up-to-date information and the system is synchronized. In this case the messages containing the actual data are sent directly from application to application. In addition some messages have to be sent to the synchronization service. These message can be relatively small compared to the actual data messages. We call this option a *distributed parallel synchronization mechanism*, even though there is still one place responsible for the synchronization.

A third method to synchronize the services is to again have a central synchronization service. In this method the services send a request to perform the next event to the central service. These events are time-stamped by the service. The central synchronization service keeps a list of all request which is ordered by time-stamp. The central service grants permission to the service with the smallest time stamp event to perform its event and send its messages. After completion the service sends the next time-stamped event request. The services are thus able to perform their request after each other thus keeping the system synchronized. We call this method *distributed serial synchronization mechanism*.

An extension of the synchronization mechanism is to make it real-time, in the way we described it above. This can be done by setting a step time length (for example 50 milliseconds). The synchronization component needs to wait with sending a ‘go’ until the time step length is reached. If the system is done sooner than the time length of the step (for instance 39 milliseconds) it has to wait until it reaches the step time length (in this case 11 milliseconds). Variation can then automatically be made to for example one week in-game time is one minute in real time, depending on the type of game that is played. The important aspect of the time-synchronization is to make sure each in-game step has equal real-time length. This condition can be stretched a little by having a catch-up mechanism to make up for lost time. An example catch-up mechanism is explained in chapter 5.

2.1.3 Causality

The concept of causality is about having an event that is the cause of another event. In a simulation/gaming environment we want to achieve strict chronological ordering of these cause events and the effects of them. If there is an event in one service that triggers a message to another service, the other service must receive and process the message before it does the calculations that are dependent on the data in the message.

2.2 Railway Gaming suite

The concept of gaming simulation suites is very broad and can be applied to many different areas. For this thesis report we had the opportunity to use a gaming simulation suite as an case study. This gaming simulation suite is called the Railway Gaming Suite (RGS). The RGS is built as part of a collaboration between the Delft University of Technology and ProRail. At the start of this thesis project the RGS was at an early stage in its development. No decision had been made on what kind of architecture will be used. Research done for this report is part of the study to find an appropriate architecture for the RGS. It is used for the first work package of the collaboration. The project work packages are discussed in section 2.2.1.

The main purpose of the RGS is to help ProRail with decision making and training. To do this the Railway Gaming Suite couples simulators and games together to provide a coherent picture of a research area. The games gives ProRail the ability to interact with the system and play-test new situations. The outcomes of the play-tests can influence the decisions ProRail has to make.

Several games are already under development during this thesis project, like a traffic controller game and a train driver game. These games are currently directly coupled to a simulator and played as single player games. See section 2.2.3 for more info on the current state of the RGS games and simulators. This section gives some background information on the Railway Gaming Suite project. First a quick overview of the organization is given and then the goals and mission statements of the project.

2.2.1 Project Organisation

The collaboration of ProRail and the TUDelft takes place between five departments within these organisations. Within ProRail three departments are involved in the project: (1) Traffic Control, (2) Innovation and (3) Capacity Management. At the TUDelft two departments are involved within the faculty of Technology, Policy and Management (TPM): (1) Systems Engineering and (2) Policy, Organization, Law and Gaming.

A steering committee is formed consisting of representatives of all five departments. They monitor the progress and results and look at the scientific consolidation and practical validation of the results. If there are differences in opinion the steering committee mediates and searches for reasonable and practical solutions. Half-yearly an evaluation of the results is performed. The steering committee decides on the continuation or adjustment of the project based on the evaluation. The steering committee is supported by a number of consultants within ProRail.

The project is divided in ten work packages. In the work packages (see below) concrete work is performed. Each work package has a project leader who is responsible for preparing a concise plan, in which at least the problem, the objective, the deliverables, the staffing, the planning and the budget are specified. For each work package a plan is presented to the appropriate members of the steering committee. ProRail must give a written approval before a work package is started.

The entire project is coordinated by a project leader of the TUDelft. His responsibilities are to support the sub project leaders and report the committee concerning the consistency and developments in the project. The project leader has a sparring partner from ProRail to coordinate with and build bridges between work packages. Each work package gets at least one contact person within ProRail to help with smooth and proper implementation of the work package. A work package leader gives substantive guidance, performs his own research activities, consults with ProRail and directs a team of TUDelft researchers. The ten work packages are:

1. System architecture of the Gaming Suite
2. PRL (PRoces Leiding) Gaming Module
3. Train Driver Gaming Module
4. Implementation connection between gaming modules and simulations
5. Scenario editor
6. Decision making with the Gaming Suite
7. ProRail Experience
8. Decision Enhancement Studio
9. Trail project agent-based gaming
10. Trail project Strategic Management Games

More information about the work packages can be found in the TU Delft, faculty of TPM, Internal project documentation, 2010. For this research, work package 1 is important. In this work package different architectures are researched and compared to each other. The goal is to find an architecture that supports the connection of the games and simulators of the Railway Gaming Suite.

2.2.2 Goals and Problem Statements of the Railway Gaming Suite

There are a number of problem statements that have led to the cooperation between the TUDelft and ProRail. The problem statements and goals are defined in the TU Delft, faculty of TPM, Internal project documentation, 2010.

- ProRail established that it needs to increase the capacity utilization of the Dutch railway network in order to meet the increasing demand for goods and passengers transportation. By planning and distributing differently, the available capacity can be significantly increased without extending the network itself. The program ‘Ruimte op de Rails’ aims to achieve a 50% capacity increase by 2020. To make this possible a number of solutions and innovative projects are defined.

- The development, realization and implementation of those solutions and innovative projects can be described as a complex multi-actor problem. This includes: 1. there are complex and dynamic interdependencies between the technical-physical parts of the system on the rail network, resulting in a high degree of uncertainty about the (un)wanted (medium term) impact of certain operational or policy measures, like ‘spoorboekloos rijden’; 2. many interdependent actors (from directors to driver) are involved, often with different opinions, interests, resources and strategies. The outcome of the strategic ‘game’ between these players is uncertain.

- Through computer simulations more insight can be obtained into the (un)desired effects of alternative measures for increasing the capacity which can take into account the dynamic interdependencies between physical and technical systems.

- Through gaming the different actors can experience the (un)desired effects of the measures in the technical and physical system themselves (interactive computer simulation, computer game) but also try out and discuss the various management strategies and decisions (role play, social or political simulation).

- This can have many positive effects such as awareness, understanding of the complexity (cause and effect relationships), readiness to change / acceptance, cooperation and coordination, improved quality of decision making, etc.

- The validation of the possible effects of Gaming and Simulation is subject of academic and applied research.

The Railway Gaming Suite is being developed to help with these problem statements. The goals of the RGS project are split into two domains, general goals and scientific goals. The general goals are:

- After the RGS is developed ProRail has a gaming suite that can be used along with existing simulation software.
- The RGS makes it possible for ProRail to set up a simulation gaming session without external help.
- The RGS makes it possible to:
  - Play-test new scenarios with existing infrastructure, tools and procedures.
  - Play-test scenarios with new infrastructure, tools and procedures.
  - Train personnel.
- The RGS will be especially applied to two domains within ProRail:
  - VL-domain (traffic control): Testing of practicability of new schedules and control concepts.
  - CM-domain (capacity management): Pre-testing capabilities for solution of conflicts in the execution. Use of the RGS as a design support suite.
- Looking at possibilities of integration with existing software, both software owned by ProRail and external software.

The scientific goals are:

- Finding out in what way gaming and simulation as a method can contribute to the analysis, innovation and training with respect to problems in a socio-technical multi-actor environment, especially rail-based infrastructures.
- This is consistent with the Gaming Hot Spot in the Next Generation Infra (NGI) research and project ”Serious Gaming for Infrastructure Design, Management and Training.” It touches on the themes Flexible Infrastructures and Understanding Complex Networks when it comes to the goals of the gaming sessions.
- In turn this contributes to the valuation of research into methods of gaming and simulation for large scale infrastructures. Outcomes and lessons on capacity management and flexibility are made available to other (international) infrastructures.
- The TU Delft aims to publish the results at conferences on game methodology (International Simulation and Gaming Association), decision analysis, training / learning, rail-specific conferences (TBA), conferences and Infrastructure (NGI).
- Much of the work within this project will be executed by two PhD students one within the Systems Engineering department and one within the department of Policy, Organisation, Law and Gaming. The work of PhD students will contribute to the scientific questions of the two chairs and will be complementary but not overlapping.

2.2.3 Games and Simulators of the Railway Gaming Suite

At the start of this chapter we stated the RGS is already in development. Sessions have been even played with a traffic controller game that is connected to a simulator. The game is called PRL and the simulator is called FRISO. In the PRL game the player takes on the role of a traffic controller. The game implementation is designed to copy the role of traffic controller as realistic as possible. The FRISO simulator was already in development when the RGS project started. It provides the game with infrastructure information and time-tables. The development and maintenance of FRISO is not done at the TU Delft. The PRL game development is done by the TU Delft.

In the past session have been played where PRL and FRISO where connected with HLA. In the current version PRL and FRISO are directly connected to each other. Another game which is at the start of development is a train driver game. Here the player takes the role of a train driver and has to drive a train through a visual 3D environment. Currently there are no multi-player possibilities for the RGS.

2.3 Summary

This chapter gives an introduction to the concept of a gaming simulation suite. It gives the definition of the term and gives some background information on it. Furthermore some terms used in the rest of the thesis are explained. Next an high level overview is given of the Railway Gaming Suite project. This is an example gaming simulation suite that is used throughout the rest of the report. In the next chapter we go deeper into the requirements of an architecture that facilitates the multi-player aspect of a gaming suite. A method called Architectural Trade-off Analysis Method is used to acquire the requirements for the Railway Gaming Suite.

Chapter 3

Trade-off analysis of the Railway Gaming Suite

In the previous chapter we talked about the Railway Gaming Suite as an example of a gaming simulation suite. We use the RGS to determine the important aspects of a gaming simulation suite. In this chapter we will to answer the research question:

- **RQ1** What are the architectural requirements to determine the suitability of gaming simulation suites?

Before we can do this we need a method to determine the requirements as stated in sub question:

- **RQ1.2** What is a good method to determine architectural requirements of a system?

The method we use to do this is developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in Pittsburg, USA. The method is called Architectural Trade-off Analysis Method(ATAM). In this chapter we will give an introduction into the ATAM. After which we describe how it was performed on the Railway Gaming Suite and what the results are. The requirements we gather from performing the ATAM will later be used to evaluate the three architectures, HLA, FAMAS en SOA.

3.1 The Architecture Trade-off Analysis Method

This section in based on the report written by Kazman et al.[21] in which they present the Architectural Trade-off Analysis Method. Over the past several years, the Software Engineering Institute (SEI^{SM}) has developed the Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}) and validated its usefulness in practice [21][13]. Below we give an overview of the ATAM, its purpose and execution steps.

3.1.1 Purpose

The purpose of the ATAM is to assess the consequences of architectural decisions in the light of quality attribute requirements. During the process of the ATAM risks, sensitivity points and trade-off points are identified and recorded. *Risks* are architecturally important decisions that have not been made or of which the consequences are not fully understood. *Sensitivity points* are the architectural parameters that have a high correlation with a quality attribute. *Trade-off points* are sensitivity points that affect more then one quality attribute in a different way. This means the outcome does not give a precise analysis of measurable data, such as calculation times. Since the ATAM is mostly done early in the development process there is not enough information to get this data. At this stage it is more important to understand what the impact on the system is for each attribute.

To be able to perform an ATAM evaluation there must be a specification of a current or possible future architecture for the system. Furthermore the quality attributes should be stated. As mentioned above often the quality attributes are not precisely defined. Using the ATAM should make these clear by:

- eliciting and refining precise statements of the architecture's driving quality attribute requirements;
- eliciting and refining precise statements of the architecture's design decisions;
- using these to evaluate the architectural design decisions to determine if they satisfactorily address the quality requirements.

3.1.2 Steps of ATAM

There are nine steps that provide a guideline for the ATAM. The steps are numbered, but this does not mean they must be taken strictly in this order. For example sometimes steps are skipped, or some steps are iterated over a couple of times.

1. Present the ATAM.
   The evaluation team gives a presentation of the process of the ATAM. Before the evaluation is started it is important for all stakeholders that are involved with the process to know what to expect and what kind of methods and techniques are used.

2. Present business drivers.
   The project manager gives a short presentation with a system overview from a business perspective. In this presentation the most important functional requirements are given, as well as the business goals and context, the major stakeholders, the architectural drivers and the technical, managerial, economic and political constraints.

3. Present the architecture.
   In this step the architectural team explains the existing or future architecture to the other stakeholders. The information in this presentation depends on the available information to the architects and the amount of time that is available for the presentation.

4. Identify architectural approaches.
   The architectural team identifies the architectural approaches that best represent the highest priority quality attributes.

5. Generate quality attribute utility tree.
   The evaluation team (the team performing the ATAM), architectural team, manager and customer representatives work together to identify, prioritize and refine the most important quality attribute goals. In this step the participants also come up with scenarios describing the system and place them in the utility tree. The utility tree is used as a guide during the rest of the ATAM. The utility tree concept is explained in section 3.1.3.

6. Analyse architectural approaches.
   Using the architectural approaches found in step 4 and the utility tree of step 5 a new list is made. This list contains for each utility sub-factor in the utility tree the associated risks, sensitivity points, trade-off points and architectural approaches that respond to it. This step gives an overview of the most important aspects of the entire architecture, the architectural decisions and a list of risks, sensitivity points and trade-off points.

7. Brainstorm and prioritize scenarios.
   With as many of the stakeholders as possible scenarios are generated. When the scenarios are created, they must be prioritized. The list of prioritized scenarios is then compared to the utility tree. If there are differences they need to be clarified and explained. This can mean a scenario might need to be better explained or the priorities can simply be changed. The utility tree is made mostly by the architectural team and the development team. Therefore the criteria for the prioritization of the utility tree can be different from the list of scenarios made by the stakeholders. This comparison makes the needs of the stakeholders clear to the architectural team. The utility tree is adjusted with the information of this step in mind and used in further steps of the ATAM. According to the prioritization only the high priority scenarios are used in the next steps.

8. Analyse architectural approaches.
   Using the new utility tree from step 7 a new mapping is made. This is done in the same fashion as in step 6. This step is done to test the mapping from step 6. In an ideal situation no new information is uncovered during this step. If this is not the case, steps 4 to 6 should be done again, until no more new information is gained.

9. Present results.
   The output of the ATAM is presented to the stakeholders. This is usually done with a verbal presentation, supported by a written report.

Two main concept used in the ATAM are scenarios and the utility tree. Therefore an short description of these concepts is given below. The Architectural Trade-off Analysis Method is usually carried out in two phases. The first phase is focused on the organization and the architecture. Steps 1 through 6 are handled in this phase. The second phase is stakeholder-centric and is meant to verify the results from the first phase. In this phase the rest of the steps are handled.

3.1.3 Scenarios and Utility Tree

A *scenario* describes the stimulus, response and environment of an interaction [5]. They are used to precisely elicit the quality goals. It is often the case that the quality attribute requirements are ambiguous or vague. In a scenario it can be made clear what the stakeholder means with them. Each of the stakeholders can describe how they would use the system. For Example:

> The system provides a virtual gaming site that includes online, offsite, and multiple games running simultaneously and that allows participants to play from their own command centers. [20]

The *utility tree* is a central structure in the ATAM process. An initial version is created during step 4 and used throughout the other steps. A utility tree is used to connect the scenarios to the quality attributes. At the root of the tree there is either the label 'utility' or the root is left out. The second level consist of the quality attributes, such as availability and extendibility. The quality attributes are further subdivided in more specific attributes. Finally the leaves of the tree contain the scenarios. Figure 3.1 shows the concept of utility trees. Utility trees help to prioritize the quality goals and make them more concrete.

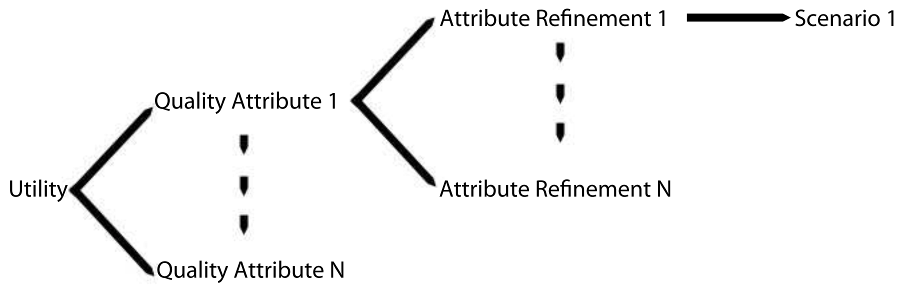


Figure 3.1: Utility Tree Concept

3.2 Performing the ATAM

When we performed the ATAM evaluation the development of the Railway Gaming Suite was in a very early state. No real decision had been made on which type of architecture was to be used. The ATAM was part of the process of finding the right architecture. Therefore there was not a real focus on a specific architecture during the evaluation itself. Since this thesis has a focus on SOAs, this did come up during the initial meeting. The steps and phases were used as a guideline, but not exactly followed step by step. The next sections give an overview of what was during the two phases. In section 3.3 we will give the outcomes of the phases.

3.2.1 Phase One

For the first phase a meeting was arranged with the two TU Delft project leaders of the Railway Gaming Suite. In this meeting steps 1, 2, 3 and 5 of the ATAM were performed. In preparation of the meeting some information was sent to the project leaders containing a short overview of the ATAM and some information on business goals as they are used in the ATAM. This was done to speed up the first steps during the meeting and to make the participants familiar with the process.

At the start of the meeting the method was further explained to the project leaders. After this one of them presented an overview of the system. A short presentation was given about service oriented architectures as an approach for the Railway Gaming Suite. Some example scenarios were created beforehand and used during the meeting as a stimulus for the process of generating quality attributes for the utility tree. An example scenario is:

> New service types (new player roles) must be inserted into the system without extensive changes/work on the existing architecture.

The example scenarios were considered during the meeting and new ones constructed. They were categorized in eight quality attributes; performance, flexibility, system constraints, reliability, ease of use, maintainability, extendibility and functionality. From the start it was clear that performance and extendibility were two of the most important quality attributes. The next meeting was planned to be with some of the stakeholders from ProRail as well as the TU Delft project leaders.

3.2.2 Phase Two

After the meeting of the first phase the results were processed to be used in the second phase. The main event of this phase was a meeting with the project leaders from the TU Delft and five stakeholders from ProRail. In preparation of this meeting the stakeholders were asked to come up with some use cases. In order to guide the construction of use cases some questions were composed together with the TU Delft project leaders:

- For which (future) ProRail project could the Railway Gaming Suite be used?
- What would be played/simulated in the gaming session?
- Who are involved in the gaming session?
- Which simulators/applications would be used in the gaming session?
- What results are expected from the gaming session?
- What is the set up of the gaming session?

A complete overview of the answers to these questions is shown in Appendix A. During the meeting the collected use cases were discussed and clarified. Some additional questions were asked for each of the use cases:

- With how many players would a session be played? (10-100-1000)
- Will there be new players joining a running session?
- Will there be new player roles joining a running session?
- How long will a session take? (hours-days)

- Where will a gaming session be held? (single room-building-different locations)
- How much time will there be between playing sessions? (minutes-hours-days)

In addition to these questions about each use case, some general questions where asked:

- How many different applications should be connected to the RGS? (10-100)
- Could you come up with a project with more or less players? (for example, zero or one player)
- What kind of timing should be supported for the scenarios? (continuous(real-time)-turn based-as fast as possible)
- How often will the RGS be used?(weekly bases-monthly bases)
- How many preparation time is there for each scenario?
- Should it be possible to run multiple scenarios at the same time?
- Who is responsible for connecting new simulators and applications?
- Who is responsible for preparing a gaming session?
- Will there be spreading of sensitive information during a gaming session?

The meeting lasted for two hours. Therefore the questions were asked to get as much information as possible without going into an elaborate brainstorm session as is custom in this phase. The questions were designed to get more insight in the quality attributes from phase one. With the results of the meeting we could construct a new list of quality attributes; extendibility, performance, consistency, availability, flexibility, usability and maintainability. This list of quality attributes was constructed together with other RGS project members of the TU Delft research team. In the next section these quality attributes are further explained. The next section also provides the other results of the two phases, the utility tree, analysis of a couple of use cases and description of the sensitivity points of the system.

3.3 Results of the ATAM

The previous section handled the process of performing the ATAM. The meetings that were held in the two phases were described. In this section we look at the final results of the meetings. First the quality attributes are stated and explained. Then the constructed utility tree is shown including the scenarios that were constructed. After which the use cases from the second phase are analysed. Finally the sensitivity points of the system are described.

3.3.1 Quality Attributes

Quality attributes are a core concept of the ATAM. As discussed in section 3.1.1 one of the purposes of the ATAM is ‘eliciting and refining precise statements of the architecture’s driving quality attribute requirements’. From the meetings in the two phases we could extract the seven driving quality attributes for the Railway Gaming Suite project. We use the RGS as a case study to get some insight into the requirements of gaming simulation suites in general. During the evaluation this was taken into account. In this section we will give a short explanation of these quality attributes the way they are meant for this project as well as explain their importance for a gaming simulation suite. We start with the three most important attributes; performance, extendibility and consistency.

- Performance: The architecture must be able to meet the real-time requirements of the RGS. In any gaming simulation suite it is important to minimize the lag caused by communication middleware.
- Extendibility: The ability to extend the system with new functionality and new games and simulators. As we stated in chapter 2 a suite is a collection of applications. Since simulation gaming is targeted at training and decision making this makes it an application domain that is sensitive to changing needs. Thus the extension of the collection is important for gaming simulation suites.
- Consistency: The RGS must be able to run in different time modes. Important parts of consistency are causality and synchronization. As stated in chapter 2 these are important aspect of simulation gaming.

Even though performance, extendibility and consistency are the main quality attributes for the RGS, there are some other attributes that are important. These are availability, flexibility, usability and maintainability.

- Availability: Errors should be handled by either the system or the facilitator, so a session can run for several hours. Error free execution is important for any software system, gaming simulation suites are no exception.
- Flexibility: Ability to connect new players on the fly. Certain scenarios can require players to enter a session at a later time or only in case of certain events.
- Usability: The RGS must be easily reconfigured to play new scenarios and sessions. The many possibilities of using a gaming simulation suite makes it important to set up new configuration without to much effort.
- Maintainability: Professional support to maintain the RGS. For any software system it is important to have good documentation on the use and workings of the system.

These quality attributes are primarily based on the RGS. In another ATAM evaluation of a simulation gaming suite, the Wargame 2000 System [20], similar quality attributes were exposed. They are however named differently.

The seven quality attributes are further refined into more specific requirements. The next section shows the utility tree that is constructed with these quality attributes with their refinements.

3.3.2 Utility tree

The utility tree for this project was constructed together with one of the researchers of the systems engineering department at the faculty of Technology, Policy and Management working on the RGS project. The refinements of the eight quality attributes are shown and scenarios associated with them are given.

Utility Tree		
Quality Attribute	Attribute Refinement	Scenario
Performance	Real-time play	Run a gaming session with 50 players while keeping it in real-time
	Redundancy	Message publication is dependant on the scenario, only useful information is published by a component
Extendibility	New applications	New project requires new application to be connected to the system without to much change to overall system
	New simulation components	The ability to extend the architecture with new simulation specific components
	New ontology	The ontology can be extended by a simulation expert to contain more data types
Consistency	Time paradigms	A gaming session can run in discrete mode
		A gaming session can run in continuous mode (real-time, slower or faster)
		A gaming session can be paused
	Causality	Events in the system are strictly chronological
	Synchronization	Events in the system are synchronized in all applications
Availability	Session Duration	Gaming sessions can take several hours; system needs to stay up the entire time
	Failure Detection	Logical faults can be detected at the architectural level
Flexibility	Support new players	During a running gaming session new players can join the game
Usability	Session-to-session	Reset the gaming environment to play a new session within 5 minutes
	Scenario set-up	A new scenario can be set up in one week

Maintainability	Documentation	ProRail will maintain the architecture
		Extension of the RGS is done by ProRail

Table 3.1: Utility Tree

3.3.3 Use Case analysis

In the second phase of the ATAM a number of use cases were created by the stakeholders. In this section we will analyse some of them by looking at the quality attributes that are important. From the questions we asked about the use cases we can find the risks that are involved with the use cases. With the risks we find during the analysis we can identify the sensitivity points of the system. These are discussed in the next section.

Scenario 1: Emergency situation Every Ten Minutes A Train (ETMET) 2015

> The situation is that during rush hour in 2015 with a timetable ETMET a disaster happens. During the game session we look at how the various actors involved in the “product” train perform their own work and where conflicts in the execution arise.

The most important aspects of this scenario are; many players with different roles, real-time applications (train driver, traffic controller), consistency between applications. The three most important quality attributes supported by this use case are; *performance*, *consistency* and *flexibility*.

Performance is affected due to the real-time requirements of the system. Some player roles in this case are strictly real-time. For example when a train driver drives from one station to another at a certain speed, the time it takes to get there should be the same as in real-time. Another aspect is the redundancy of the messages. In a system with many services, the number of messages should be as few as possible in order to prevent congestion in the network. Risks involved with this are that when there are to many players the system gets clogged and is unable to meet the real-time requirements.

The *consistency* is very important in a session like this because the player roles are closely connected to each other. When a traffic controller sets a specific route for a train to go, the switches in the train driver games should be set accordingly before the train driver arrives at the switch. This means the events should be strictly chronological and synchronized across the system. The consequence is that the games need to ‘wait’ for each other. When one game is not done with calculations other games can not continue as well.

In this case there are a lot of different players with different roles, which translates to different services/games connected to each other. This supports the *flexibility* quality attribute. The consequence of this is that clear joining protocols must be defined. The risk that emerges is that messages can get lost if the protocol is not implemented well. Another risk is that the entire session can lag in order for a new player to join, depending on the implementation.

Scenario 2: Bridge opening possibilities 'Vechtbrug'

> A situation at the 'Vechtbrug' with fixed and flexible bridge opening times. The game sessions examines the differences for the traffic controller between the current protocol and the situation with flexible bridge-times.

This use case is concerned with multiple sessions running different versions of the same scenario. The main quality attributes are the *usability* and *extendibility* of the system.

For ProRail this is a typical case where they want to use the RGS very soon after they encounter the problem. Ideally all services for such a scenario are ready and it can be played as soon as possible. Otherwise it should not take to long (one person month) to create new services to connect the required games or simulators. When all services are ready, setting up a new scenario with them should not take more then one person week. When the scenarios are set, it should not take more then a couple of minutes to switch between them. All *usability* refinements are supported in this use case. Only part of these requirements can be solved by the architecture however, like the ability to restart the services. The other part is dependent on the applications themselves, how easily they can be reconfigured for a new scenario.

Connecting new services is part of the *extendibility* of the system. The creation of new services should be as generic as possible. New services could also mean new message types to send across the system (extending the ontology). A risk that emerges from this could be that a new service provides a lot of new message types, used by existing services. These services should all be updated then to support these new messages. This could potentially mean a lot of work.

Scenario 3: Traffic control in Japanese style

> Two or more traffic controllers are leading the rush hour traffic through a congested corridor. Only 'stop trains' and 'intercity trains' are driving here. They have new means of traffic monitoring for this. Setting routes is done completely automatically, based on current plans. When a disturbance occurs, first the plan is updated and then given back to the machines for setting the routes.

We analyse this use case so the other quality attributes, *availability* and *maintainability*, can be handled as well. They are also supported in the other use cases, but were not specifically discussed.

Since most sessions can take several hours it is important the system stays up the entire time. When an error occurs, it should either be handled by the facilitator or otherwise taken care off by the system. This supports the *availability* requirement.

ProRail will eventually maintain the RGS. Therefore documentation on how to do this needs to be of good quality. The quality of the documentation determines part of *maintainability*.

3.3.4 Sensitivity and Trade-off points

During the use case analysis we uncovered five important sensitivity points. A sensitivity point is an architectural decision that has a strong effect on a particular quality attribute. We identified performance and extendibility as two most important quality attributes at the start of the ATAM. The sensitivity points are related to these quality attributes. The use case analysis showed the importance of other requirements, such as the synchronization and the session duration. Implementation of these other requirements either directly affects the two main quality attributes or there is a trade-off in how strict they are implemented.

The first sensitivity point is:

> The performance is sensitive to implementation of the architecture.

Implementation decisions on how to handle network deployment and message transmission, for example, have a direct influence on the number and size of messages on the network. The delivery speed of messages is dependent on the transmission protocol. All these implementation decision have an effect on the performance of the architecture. The types of scenarios that will be played with the Railway Gaming Suite require it to be real-time with player numbers up to fifty.

The second sensitivity point is:

> Making the system synchronized and keeping strict causality comes at a cost of performance.

Synchronizing the services with each other requires additional messages to be send across the network. This has a direct effect on the performance. Another effect is that service need to wait on each other before continuing, so the system is as slow as is slowest participant. The synchronization is however required in this system and thus is not a trade-off. The causality requirement is also a sensitivity point that affects the system in the same way as the synchronization.

The third sensitivity point is:

> Increasing reliability comes at a cost of performance.

Making sure the system stays up during sessions of a couple of hours requires monitoring mechanisms to check on all the applications. This means more messages being sent across the network. Depending on the implementation of the monitoring mechanism this has more or less effect on the performance. Here a trade-off arises on how reliable the systems should be while keeping the performance in mind.

The fourth sensitivity point we identify here is:

> Adding new players during a running session can disrupt the real-time requirement.

The ability to join a running session has the consequence that a new service needs to be brought up-to date with the current state of the system. Doing this potentially takes more time then there is available while keeping the system running in real-time. A trade-off must be made on how strict the real-time requirement must hold in the event of a new player joining.

The last sensitivity point we identified is:

> Ease of adding new services, components or ontology is sensitive to the quality of the documentation.

As stated above the Railway Gaming Suite will be maintained and extended by ProRail. Even though this will be done by the technical staff (and programming skills should not be a problem) the implementation details and the inner workings of the architecture are not common knowledge and most thus be well documented by the system architects.

3.4 Summary

In this chapter we have given an overview of the Architectural Trade-off Analysis Methods and shown how it was performed for the Railway Gaming Suite. We started with giving the goal of the ATAM. Then we explained the steps and phases in which the ATAM is performed in general and discussed the most important terms, utility tree and scenario.

After the introduction of the method we continued to show how it was performed for the Railway Gaming Suite. First we described how the two phases were performed, by describing the meetings that have taken place. Then the outcomes of the meetings we supplied, by means of the quality attributes, a utility tree, an analysis of some of the use cases and finally an overview of the most important sensitivity points and what trade-offs there are concerning them.

Chapter 4

Service Oriented Architecture

Service Oriented Architecture (SOA) is a paradigm for modeling a distributed software architecture. This gives us two terms that are important, namely *software architecture* and *distributed*. A definition for software architecture is given by Clements et al. [12]:

> *Software architecture* is about structural properties of a system. Structural properties can be expressed in terms of components, interrelationships, and principles and guidelines about their use.

A *distributed system* is described by Andrews[4] as

> Several autonomous entities that communicate with each other by message passing.

In this chapter we will focus on a service oriented architecture as a form of a distributed architecture. Thereby answering the research question:

- **RQ2.1** What are the principles of a service oriented architecture?

First we take a look at the principles of distributed architectures in section 4.1. Then we focus in service oriented architectures in general in 4.2 as well as some design approaches for building a SOA. Finally we give an example of what the Railway Gaming Suite could look like using SOA as the underlying architecture.

4.1 Distributed Architectures

As described above, a distributed architecture is responsible for the communication between entities or otherwise called *nodes*. The nodes are located on two or more systems (computers) and are connected to each other through a network connection. We take a look at the two important aspects of a distributed architecture that facilitate the communication between nodes across a network, namely message transmission and network deployment.

Message transmission is the way a message is sent from one node to another. There are three main techniques for this, unicasting, multicasting and broadcasting (Figure 4.1). In unicasting(a) a message is sent from the sender through the network to one specific receiver. In multicasting(b) a message is send from the sender to multiple

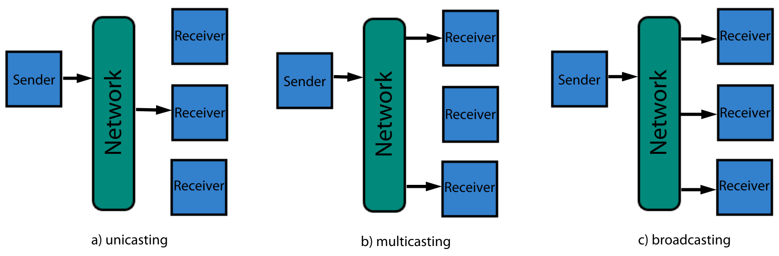


Figure 4.1: Message transmission

specific receivers. Finally in broadcasting(c) a message is sent from the sender to all applications in the network.

Network deployment is about the connection of the nodes in a network. There are two techniques for this: peer-to-peer and client-server (Figure 4.2). In peer-to-peer(a) all nodes are connected to each other and all nodes are the same. It is possible to send a message to every other node directly. In client-server(b) there is a special node (server) to which all other nodes (clients) connect. All messages are sent to the server node, which in turn sends it to the correct receiving client nodes. It is possible to construct hybrid solutions for node connection. For example a peer-to-peer server network, where each server has a number of clients attached to it. The clients still connect to a single server, but the server may need to relay a message to another server, which in turn sends it to one of its clients. Another construction could be using a 'super-server', where multiple servers are connected to a single 'super-server'. Exploring these constructions further is outside the scope of this report.

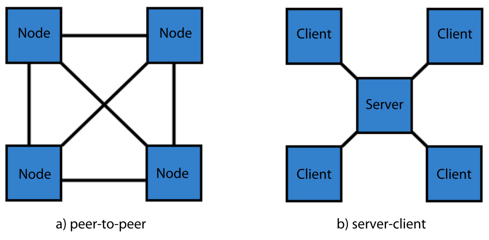


Figure 4.2: Network deployment

4.2 Service Oriented Architectures

Service-oriented architecture (SOA) is a very popular architecture paradigm for designing and developing distributed systems. SOA solutions have been created to satisfy business goals that include easy and flexible integration with legacy systems, streamlined business processes, reduced costs, innovative service to customers, and agile adaptation and reaction to opportunities and competitive threats. [7]

In this chapter we look at the high level ideas behind service oriented architectures, without going into a specific implementation. The introduction of this chapter stated that a distributed architecture connects nodes in a network to each other in order to make communication possible. In a service oriented architecture the nodes are called *services*. Because a SOA is a distributed architecture the nodes/services want to communicate with each other. The need to communicate arises when a service requires information another service can provide. In a SOA the requesting service is called a *consumer service* and the service that delivers the information a *provider service*. It is possible for a service to require and provide information, so to be a consumer as well as a provider. Before services can communicate they need to find each other. In service oriented architectures there is an additional service role, which can be used for this, called the *service broker*. Publishing services need to announce themselves to the service broker. The service broker keeps an index of what each service provides for the consumers to request. This way a consumer can find out where to get the information it needs [24].

What we see in the above description is that the notion of services is central in SOAs. Bianco et al. [7] describe the following characteristics of an ideal service in a service oriented architecture:

- A service is self-contained. The service is highly modular and can be independently deployed.
- A service is a distributed component (or collection of components). The service is available over the network and accessible through a name or locater other then the absolute network address.
- A service has a published interface. Users of the service only need to see the interface and can be oblivious to implementation details.
- A service stresses interoperability. Service users and providers can use different implementation languages and platforms.
- A service is discoverable. A special directory service allows the service to be registered, so users can look it up.
- A service is dynamically bound. A service user does not need to have the service implementation available at build time; the service is located and bound at runtime.

These are the ideal characteristics, in most cases some are missing or not fully implemented [7]. The three most important are that services are loosely coupled, self-contained and discoverable over the network. In the next subsections we go deeper into the design approaches of building a SOA implementation and how they have an impact on the overall functionality and application of the SOA.

SOA is an architectural paradigm and therefore many implementations of it are possible. In literature service oriented architecture is often directly linked to Web Services[1]. This is however one possible implementation of a SOA. Therefore in this report we do not go into the specifics of Web Services.

[1] *http://en.wikipedia.org/wiki/Web_service/*

4.2.1 Network Deployment

The SOA paradigm does not specify a strict design approach for the network deployment. In the first section we showed two approaches for network deployment. Here we will look at their impact on the system.

In a client-server model all services connect to a single server hub. This increases the flexibility, because new services only need to connect to the server and it takes care of the rest. All messages need to go through the central hub, which has a negative effect on the performance. The service broker structure described above can be implemented as a client-server model. The brokering software is often called an *Enterprise Service Bus (ESB)* in SOA solutions [7]. All services interact only with ESB, which is responsible for the data routing.

In a peer-to-peer model the services are directly connected to each other. This has a negative effect on the flexibility of the system, since for every new service a new direct connection must be implemented. On the other hand the direct communication has a positive effect on the performance. In a strict peer-to-peer implementation all services are equal. When we deviate a little from this we can implement a service broker structure in a peer-to-peer environment. We can create one special service (a service broker) which is responsible for connecting the other service to each other. This way all services need only announce themselves to the service broker, making it more flexible. After announcement there is a direct connection to the other services and data is sent directly, keeping the performance high.

4.2.2 Message Transmission

The three message transmission techniques described in section 4.1 are all possible in a SOA implementation. It depends on the functionality of the system which technique is most applicable. The unicast technique can be used when service consumers know exactly which of the providers has the information they need and information provided is always only interesting for one service. The advantage of this technique is that the number of messages is limited to the essential. The multicast technique can be used when most messages are interesting for more then one consumer. In order to make sure messages are sent only to the service consumers that want them a more complex mechanism is needed. Depending on the mechanism it is possible to send to many messages across the network, with a chance of decreasing the overall performance. The broadcast technique sends all messages to every services. This way all message are received by the consumer that needs them, but also by consumers that do not need them. The consequence is that too many messages are sent and the performance decreases.

In service oriented architectures a popular mechanism for message transition is a publish-subscribe mechanism. Which is a multicasting technique, where it is possible that the number of receiving nodes is one, some or all of the other nodes. This means it can unicast and broadcast as well. The publish-subscribe mechanism is based on the observer pattern defined by Gamma et al. [16]. Each service can publish and subscribe to messages. Services are made aware of the published information in the system and can directly subscribe to it. This way messages are only sent to receivers that want the information. Even though publish-subscribe mechanisms are often used in service oriented architectures it is not obligatory to do so.

4.2.3 Service Discovery

One of the SOA characteristics is the ability of the services to find each other. The need for this is dependent on the network deployment and message transition techniques that are chosen. In a client-server system using an ESB, services need only know how to connect to the ESB. Services can be completely unaware of each other.

In a strict peer-to-peer system all services need to be made aware of each other. This means either the services search a specific address space looking for other services, or they need to have a list of all addresses of the other services in advance. In the first option the address space should be limited (a specific IP-address range for example) otherwise it takes to long too find other services. The second option makes sure the services are able to find each other. It is however less flexible, since locations of new services that want to join the network need to be known in advance.

The less strict peer-to-peer option described above has the advantage of a single connection point, like with the ESB. Services need to announce themselves to the service broker. It will keep a registry of all service consumers and providers and send their addresses back to the newly connected service. The service can in turn connect to the other services in the system. This option has the flexibility of having a single connection point at start-up.

4.2.4 Interoperability

Another important characteristic of a service oriented architecture is that the services are self-contained. A consequence is that the services are implemented in different languages and have different object models. This characteristic of SOAs leads to mismatches in technology and messages types. The architecture should make sure this is handled correctly. In a SOA the interpretation from one object model to another is often called *adapting* and is done by what is called *adapters*. One way to do this is sending data from one service to the other by adapting it directly into the object model of the receiver. Another way is to adapt it into a intermediate object model and from the intermediate object model to the receiver object model.

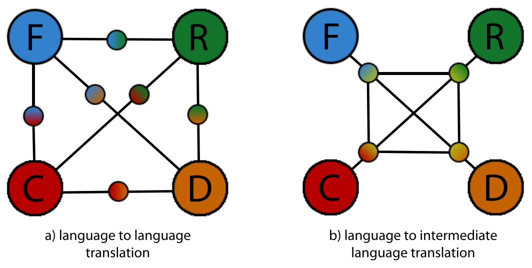

F
R
C
D
a) language to language translation
F
R
C
D
b) language to intermediate language translation

Figure 4.3: Message translation

For example four diplomats (a French diplomat(F), a Russian diplomat(R), a Chinese diplomat(C) and a Dutch diplomat(D)) want to communicate with each other (see figure 4.3). This can be done by using interpreters for each pair of diplomats. Six interpretors are needed then (F-R, F-D, F-C, R-D, R-C, C-D). As the number of diplomats increases, the number of interpreters increases with a rate of $\binom{diplomats}{2}$. Another way to do it is to introduce a new intermediate language (for example English). Now each diplomat needs one interpreter, interpreting from its own language to English and the interpreters can communicate directly to each other in English. Now as the number of diplomats increases the number interpreters increases at the same rate. Using an intermediate object model makes the system less complex and more flexible. However an extra adaptation of the data is needed, which has a negative effect on the performance.

4.3 Railway Gaming Suite

In chapter 2 we gave an overview of the Railway Gaming Suite as an example gaming simulation suite. With the description of service oriented architectures above we can construct a high level design of the Railway Gaming Suite using a SOA architecture. In figure 4.4 an example is given of some services that will be part of the RGS. We use the traffic controller game (PRL) and one of the simulators (FRISO) in this example since they are already in development. The gaming services are:

- Timing service, responsible for time management, synchronization and causality.
- Logging service, responsible for keeping track of events during sessions, like human interactions and time-tables.
- Facilitator service, responsible for providing control on the system to the facilitator.

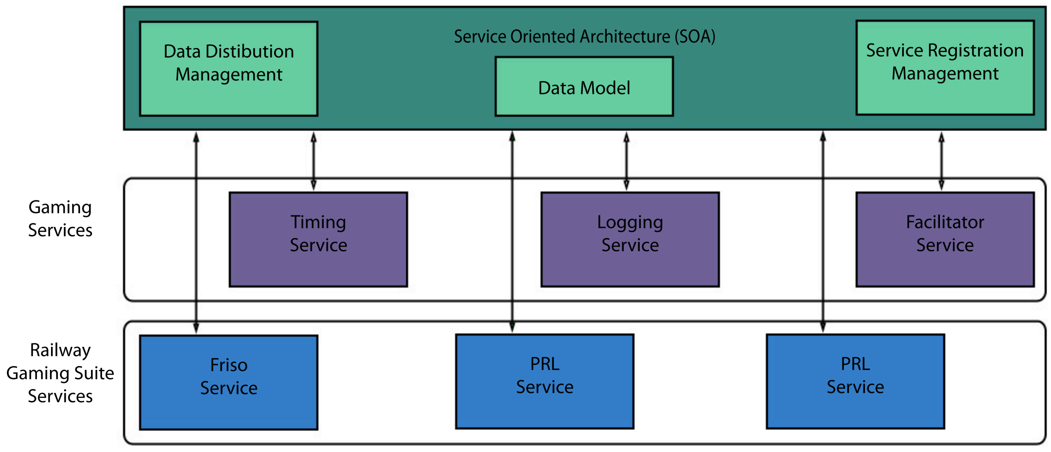

Service Oriented Architecture (SOA)
Data Distibution Management
Data Model
Service Registration Management
Gaming Services
Timing Service
Logging Service
Facilitator Service
Railway Gaming Suite Services
Friso Service
PRL Service
PRL Service

Figure 4.4: High level architectural overview of part of the RGS

The architecture has been depicted in a bus like fashion, but this is not necessarily the network deployment option for the RGS. Message transmission is handled in the data distribution management module and service discovery in the service registration

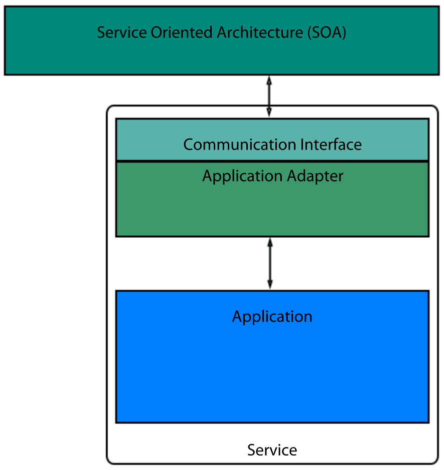


Figure 4.5: High level architectural overview of a service

management module. In the case of an intermediate model this is also part of the architecture, we call it the data model module in this case.

We can take a closer look at what a service should looks like (figure 4.5). The main components in a service are:

- The application. For example PRL see figure 4.6.
- An adapter. This could be either adapting to an intermediate object model or directly to the target object model. In the latter case the adapter should contain adaptation to all other object models.
- Communication interface. This interface takes care of the coupling to other services, the implementation of this depends on the design decisions taken.

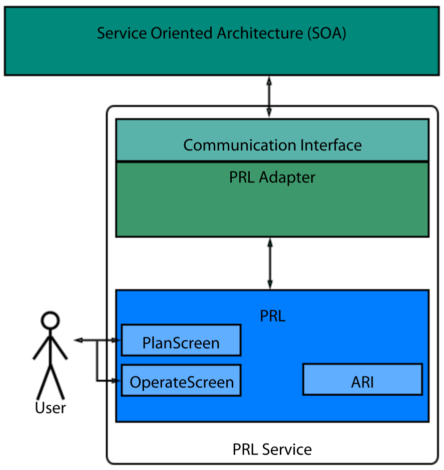


Figure 4.6: High level architectural overview of PRL Game

4.4 Summary

In this chapter we have taken a look at service oriented architectures as a form of distributed architectures in order to answer the sub question: What is a service oriented architecture? First we gave a quick look at distributed architectures in general. We handled the two main concepts of network deployment and message transmission. Then we have taken a closer look at service oriented architectures. We looked at the roles nodes can have in a SOA and then delved deeper in the concept of a service. Next we looked at different design approaches for network deployment, message transmission, service discovery and interoperability. Finally an example overview of the Railway Gaming Suite was given using the service oriented paradigm to connected the games and simulators.

Chapter 5

Service Oriented Gaming and Simulation

In this thesis we want to test whether an architecture based on the service oriented paradigm is an appropriate choice as a simulation gaming suite architecture. In chapter 7 we will evaluate this according to the quality attributes we found during the ATAM evaluation to answer the research question:

- **RQ2** How well does a service oriented architecture support the requirements of gaming simulation suites?

In order to answer this question we can look at the principles of SOA as discussed in chapter 4. The performance requirement however can be evaluated by performing tests. In order to get these test results for a SOA architecture we have implemented a prototype architecture based on the SOA principles. Implementing a prototype architecture also gives some insight in the ease of making a SOA keeping simulation gaming aspects in mind. Using this prototype to make a test scenario gives some hands-on experience with connecting applications to the architecture and the extendibility requirement. This chapter is related to the research question:

- **RQ2.2** Can we construct a prototype SOA to test performance requirements?

The architecture is designed to facilitate the gaming simulation specific functionalities, namely real-time play and system wide synchronization. In this chapter we will handle the design decisions for the prototype and the actual implementation. The name of the architecture is Service Oriented Gaming and Simulation (SOGS).

5.1 Design

The Service Oriented Gaming and Simulation (SOGS) architecture aims to provide the flexibility and high performance that is needed in gaming and simulation suites. In this section we will describe how the four main design aspect of sections 4.2.1 through 4.2.4 are handled for the SOGS architecture. Furthermore we look at the design of the gaming simulation specific functionalities of sections 2.1.1 and 2.1.2 for the SOGS package.

5.1.1 Network Deployment

In chapter 4 we have shown there are two main ways to handle the network deployment from an architectural perspective, client-server and peer-to-peer. For the SOGS architecture the peer-to-peer approach was chosen. The reason for this choice is the that the peer-to-peer approach has the potential to reduce the amount of data send across the network. In this approach it is possible to send big messages from service provider to service consumer directly without first sending it to the server and then sending it from the server to the consumer, like it would be in a client-server system.

5.1.2 Service Discovery

For SOGS a mechanism using a service broker (see section 4.2) was chosen. The role of the service broker is to connect the other services to each other. This way the services need only know the location (for example the IP address) of the service broker and it takes care of the rest. This increases the flexibility since it is not necessary for all services to be known or know each other at the start-up of the system. Announcement of a service is done by request. The ServiceBroker decides when and if a service can announce itself.

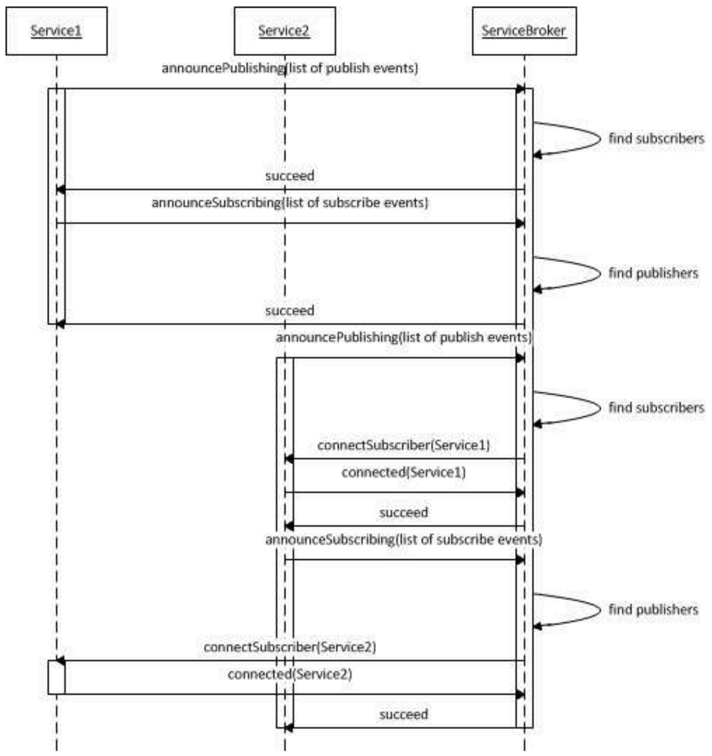

Service1
Service2
ServiceBroker
announcePublishing(list of publish events)
find subscribers
succeed
announceSubscribing(list of subscribe events)
find publishers
succeed
announcePublishing(list of publish events)
find subscribers
connectSubscriber(Service1)
connected(Service1)
succeed
announceSubscribing(list of subscribe events)
find publishers
connectSubscriber(Service2)
connected(Service2)
succeed

Figure 5.1: Announcement protocol

In figure 5.1 the announce protocol is shown for two services and the server. In this example Service1 has an event which Service2 subscribes to and the other way around. At start up Service1 announces itself to the ServiceBroker both for publishing events and subscribing events by sending a list with the publishing and subscriber events. Events are identified by event type. Service2 does the same. When Service2 announces itself the ServiceBroker finds Service1 in its list and returns this to Service2. Now Service2 has the location of Service1 to send the events to. The other way around Service1 is notified of the location of Service2 by the ServiceBroker since Service1 was in its the subscribe list. The announce messages are done by request. In the example there is no problem with the request at the service broker side, thus the announcements succeed. It could be the service broker bounces the request, or puts it on hold.

5.1.3 Message Transmission

Message transmission in SOGS is done using a publish-subscribe mechanism. In figure 5.1 it is shown that upon announcement the services are connected by events they publish and subscribe to. For each publishing event a connection is created with all subscribers of the event. This is done when a new service announces itself. After announcement it is still possible to publish more or less events or subscribe to more or less events. This is done by making a request to the ServiceBroker. The ServiceBroker decides when and if the new connections are made. The publish-subscribe mechanism reduces the number of messages send across the network to a minimum, by only sending to service consumers that requested the data.

5.1.4 Interoperability

In the SOGS architecture an intermediate language will be used. This decision is instigated by the great diversity of possible games and simulators that must be able to communicate with each other. As such, this adds to the flexibility of SOGS as the intermediate language makes that each new service only needs to be able to translate to or from this language. In reality it is more complicated, because this requires the intermediate language to contain all possible data the services want to publish. In the case of the Railway Gaming Suite, this means the intermediate language should for example be able to handle location data of the trains in a visual environment (train driver application), as well as time-table data (traffic controller application). In other words a clear ontology is required to make sure all services are able to communicate with each other. Constructing an ontology is specific to the project the architecture is used for. For SOGS a tool is designed to make the construction of new objects easier. This tool is called the SOGS Data Model Builder (see Appendix B. Extending the ontology in an existing system requires the services to update their adapters. An example adaptation is given in the implementation section on interoperability.

The overhead of translating the message twice does not outweigh the gain in flexibility of the system. Part of this is because the different applications that are connected to each other already require a message to be translated multiple times.

5.1.5 Timing and Synchronization

As stated in chapter 4 one of the most important characteristics of a service oriented system is to have loosely coupled services. Therefore SOGS has a separate service for the timing and synchronization. The synchronization mechanism ensures the data send across the network is synchronized.

One aspect that should be synchronized is global timing. The synchronization service is ideal to handle the timing as well. The synchronization works as explained in section 2.1.2. In particular we use the second method described in that section, the distributed parallel synchronization mechanism. For timing we have chosen to integrate

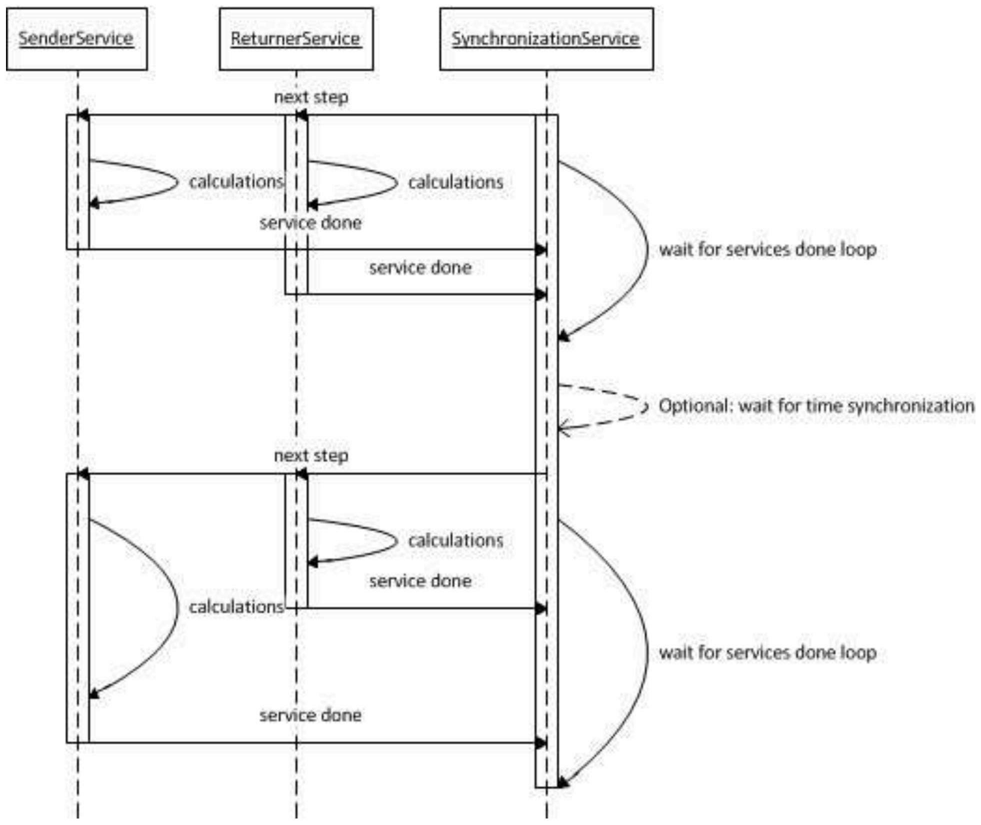


Figure 5.2: Time synchronization in SOGS.

this in the synchronization mechanism, as described in section 2.1.2. Before sending the next time step a check is made to see if the system is still running in correspondence with the wall-clock. A catch-up mechanism is part of the timing to make up for lost time. The details of the timing mechnism are explained in the implementation section.

5.1.6 Classes of SOGS

With these design decisions a class diagram of the SOGS library could be constructed, see figure 5.3. The most important classes are:

- AbstractService
- BasicService
- ServicePublisher
- ServiceSubscriber

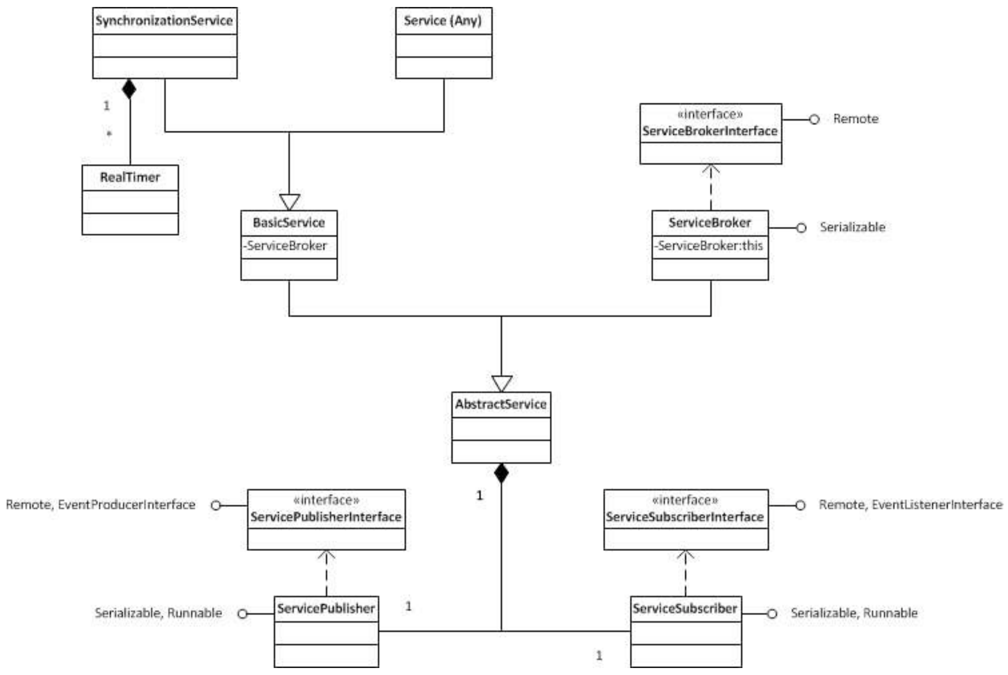


Figure 5.3: Overview of the classes in the SOGS package.

- ServiceBroker
- SynchronizationService

The first five classes are needed to connect applications to each other. The SynchronizationService is a specific application used in the synchronization and timing mechanism. For the timing mechanism the class RealTimer is used. Figure 5.3 gives a high level overview of the SOGS package. In the next sub sections we will explain the roles of the classes by functionality as discussed in the design section.

5.2 Implementation

The SOGS architecture library is implemented in Java. Java was chosen because of the familiarity of the developer with the language and use of the DSOL (distributed simulation object library) library [1] which is also written in Java. DSOL is a library that contains functionalities to build a simulation environment. For this implementation only the event system of the DSOL library was of interest.

The DSOL event system uses Java Remote Method Invocation (RMI) [2] to send messages between applications. Messages are contained in Events in this system. A DSOL Event contains an object containing an identifier for the source, an object containing the data and an EventType. The source identifier can be any Java object, for example a String. The data object can be any Java object as well. EventTypes are identifiers for the Event. All objects send with RMI have to be serializable.

[1] http://sk-3.tbm.tudelft.nl/simulation/node/4

[2] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

Next to the normal Event DSOL also contains a TimedEvent, which adds a timestamp to a normal Event. For the SOGS architecture we extended the TimedEvent to contain a unique message identifier. The new Event object is called UniqueTimedEvent. The addition of the unique message identifier could help with robustness mechanisms in future builds. The unique identifier can be used to check the delivery of a message without having to send the data again.

The DSOL event system is used to implement the publish-subscribe mechanism of the SOGS package. Publishing is done using DSOL's RemoteEventProducer and subscribing using DSOL's RemoteEventListener.

EventTypes act as identifiers. They are used to identify the events a service wants to publish or subscribe to. In SOGS we make a difference between two different EventTypes, namely EventTypes concerned with the application domain and EventTypes used by the architecture. The difference is made because we might want to add extra operations concerned with only the architectural or the non-architectural EventTypes. For example sending additional messages (containing the unique identifier and EventType) to an administrator service, to keep track of the messages sent in the system without having to send the big payload of the original message.

5.2.1 Network Deployment

The peer-to-peer approach of the SOGS architecture requires the services to communicate directly with each other. We have implemented the AbstractService class to set up the connections. AbstractService creates the classes for the publish-subscribe system, see section 5.2.2. Every service in the system needs to directly or indirectly extends the AbstractService class. The AbstractService extends the Java Thread class and defines the run loop of the service:

```
public void run()
{
  printMessage("Service " + getServiceName() + " started");
  preLoop();
  synchronized (this)
  {
    while(this.running)
    {
      try {
        this.wait();
      } catch (InterruptedException e) {
        this.errorOccured("'unauthorized interruption during run'");
      }
      inLoop();

      this.notifyPublisher();

      serviceDone();
    }
  }
  postLoop();
}
```

Implementation of the preLoop, inLoop and postLoop methods are specific to the application(s) the service connects to the system. The wait() and serviceDone() are used for the synchronization mechanism, see section 5.2.4 and figure 5.2.

5.2.2 Service Discovery and Message Transmission

As discussed in the design section we use a service broker mechanism. A class appropriately called ServiceBroker is constructed for this. The discovery mechanism itself uses a Java RMI registry at the ServiceBroker side. For the other services to find the ServiceBroker class we extend the AbstractService class. The extending class is called BasicService. It extends the AbstractService class with the functionality to find the registry of the ServiceBroker by IP address and port number and this connect to the ServiceBroker. All services (except the ServiceBroker) in the system need to extend the BasicService class. The connection of the services is done by the ServiceBroker is done as described above in the design section and further explained below.

The AbstractService creates two Threads, ServicePublisher and ServiceSubscriber, to handle the publishing an subscribing. The connections between services are made directly between the ServicePublishers and ServiceSubscribers. Each ServicePublisher and ServiceSubscriber that announces itself at the ServiceBroker is put in a list sorted by the EventTypes it publishes or subscribes to. The ServiceBroker then searches for each EventType that is published, whether there are subscribers to it and connects them. The same way the subscribers are connected to the publishers, by searching if there are publishers for the subscribe EventTypes, see below for implementation.

```
private synchronized boolean setSubscriber(
  ServiceSubscriberInterface subscriber,
  EventType eventType) throws RemoteException
{
  boolean success = false;
  if (this.publishers.containsKey(eventType)) {
    ArrayList<ServicePublisherInterface> publisherList
      = this.publishers.get(eventType);
    for (ServicePublisherInterface publisher : publisherList) {
      // if publisher is not the subscriber...
      if (!publisher.getName().equals(subscriber.getName())){
        // ... add the subscriber as a listener
               to the publisher for the specific eventType
        try
        {
          success = publisher.addListener(subscriber, eventType);
          if (!success)
          {
            break;
          }
        } catch (RemoteException remoteException)
        {
          this.errorOccured("setSubscriber:-
              remote exception when adding listener "
              + subscriber.getName() + " to service "
              + publisher.getName() + ".\n"
              + remoteException.toString());
        }
      }
    }
  }
  return success;
}
```

After initial announcement a service can request to publish and subscribe to an additional EventType or remove an EventType. The ServiceBroker takes care of the connection again. Other services can request the service broker to publish and overview of the current publishing and subscribing services in the system. This is used in the synchronization mechanism. The ServiceBroker class itself extends the AbstractService class and thus can publish and subscribe to events like any other service.

5.2.3 Interoperability

Since SOGS is written in Java and uses Java RMI to send messages back and forth, the content of the messages are Java Objects. The wrapper object in the RMI messaging system is the UniqueTimedEvent for SOGS. Within the UniqueTimedEvent the actual data is stored. UniqueTimeEvent requires this to be a Java Object again. All application specific data must be converted to Java Objects in order to send it to other applications. The actual implementation of the Java Objects is free and dependent on the data exchanged between the services. For example in the Railway Gaming Suite information about trains could be published. We can imagine we need an object called Train, with different attributes like type, maximum speed, number of passengers. Each service needs to know exactly what the train object sent from another service looks like in order to extract the information. For this we use an adapter construction as described in section 4.2.4. Within the adapter it can then extract the information from the Train object and adapt it to the application specific data. The intermediate language consists of Java Objects for SOGS. In order to make the process of creating new Java Objects for the intermediate language easier and quicker a tool was created. More information on the tool can be found in Appendix B

The adapting of messages works as follows in case of FRISO and PRL (see chapter 2). In this example FRISO want to publish a message which announces a train for a particular PPLG (this is a part of the railway network). The message identifier is 1117. This identifier is used by FRISO to indicate an ‘announce train’ message. It is also used in the data model for the EventTypes. In the SOGS architecture the FRISO XML message is adapter to the intermediate data model object and upon arrival by the PRL-service from the intermediate data model object to PRL objects. PRL is written in Java and uses Java classes for its internal objects. In this case the timing is in milliseconds and the same in all applications, so no adaptation is needed. Otherwise the adapter needs to take care of this as well. The Jave code is written in pseudo code to keep it simple. In appendix B an example is given with the complete Java code of an example SOGS intermediate data model MessageType.

The FRISO-service gets the following data from FRISO. This example is taken from a FRISO-PRL communication log.

```
<ROOT>
  <HEADER>
    <MESSAGE_ID>1117</MESSAGE_ID>
    <TIME>10407</TIME>
  </HEADER>
  <MESSAGE>
    <PPLG_CODE>ASDZ</PPLG_CODE>
    <TRAIN_NAME>
      <TRAIN_NAME>1511-H-2</TRAIN_NAME>
      <TIMETABLE_REP>1</TIMETABLE_REP>
     </TRAIN_NAME>
  </MESSAGE>
</ROOT>
```

The FRISO-adapter takes this and transforms it to the intermediate data model. The data is sent as a UniqueTimeEvent. The MESSAGE_ID and TIME are used for the EventType and timestamp. The intermediate data model uses Java objects to store the information of the message. With the SOGS Data Model Builder a MessageType and a DataType are constructed to store the data. The MessageType looks as follows:

```
public class AnnounceTrain
  String pplgCode;
  TrainName trainName;
```

In this case the new data type is TrainName:

```
public class TrainName
  String trainName;
  int timeTableRep;
```

The UniqueTimeEvent is constructed as follows:

```
AnnounceTrain announceTrain = new AnnounceTrain("ASDZ", "1511-H-2",1);

UniqueTimeEvent(1117,FRISO-Service,announceTrain,10407,UniqueIdentifier);
```

After adaptation this UniqueTimeEvent is published by the FRISO-service. The PRL-service is subscribed to this event and will receive it. The PRL-adapter takes the UniqueTimeEvent and adapts it for use by PRL. First it gets the data from the AnnounceTrain object and adapts it to objects as PRL uses them:

```
String pplgCode = announceTrain.pplgCode;
```

and

```
TrainID trainID = new TrainID(announceTrain.trainName.trainName,
  announceTrain.trainName.timeTableRep);
```

Then using the EventType the data is given by the PRL-service to the correct method in PRL:

```
announceTrain(PplgCode, trainID);
```

5.2.4 Timing and Synchronization

Timing and Synchronization is handled by a simple service, the SynchronizationService which extends the BasicService. It is therefore connected to the other services by the ServiceBroker and can publish and subscribe to EventTypes. The SynchronizationService follows the synchronization mechanism from figure 5.2. It publishes a NEXT_TIME_STEP message to the system. This allows all services to start performing their calculations. It then waits until all services have transmitted they are done with their calculations, They do this by publishing a SERVICE_DONE message. As soon as all services are done the SynchronizationService can send a new NEXT_TIME_STEP message. Since the SynchronizationService regulates the synchronization it has a different run method then the AbstractService which is constructed to integrate the synchronization, see below.

```
public void run()
{
  preLoop();
  while(this.running)
  {
    inLoop();
  }
  postLoop();
}
```

The inLoop handles the actual synchronization:

```
public void inLoop()
{
  try
  {
    long now = System.currentTimeMillis();
    setCheckList();

    nextTimeStep();

    checkAllServicesDone();

    if (this.timed)
    {
      RealTimer.getInstance().nextTimeStep(now);
    }

  } catch (InterruptedException e)
  {
    System.out.println("unexpected close!");
    e.printStackTrace();
  }
}
```

To synchronize this mechanism to wall-clock time it can use the RealTimer class. The RealTimer check whether it is time to go to the next time step. If not it performs a Thread.sleep() for as long as there is time remaining in the time step. In case the step took more time then it was allowed to use this delay is stored in a backlog variable and the RealTimer does not perform a sleep so the NEXT_TIME_STEP message is sent immediately. The backlog is used to catch up the lost time in the next time steps. This is further explained below.

Upon initialization of the RealTimer a begin time is set, a begin game time, the length of each time step and a time factor. The begin time is the current system time (in milliseconds) at the initialization of the RealTimer. It is used to correct the error of the slightly inaccurate Thread.sleep() method. Each time step the current system time is checked against the time the RealTimer expects it to be. The error is added to the timer backlog.

```
public long nextTimeStep(long beginTime) throws InterruptedException
{
  long used = System.currentTimeMillis() - beginTime;
  this.backlog += System.currentTimeMillis()
                  - this.startTime
                  - this.currentTime
                  - used
                  - this.backlog;
  long delay = Math.round(this.timeStep/this.timeFactor - used);

  this.currentTime += this.timeStep;

  if (delay >= 0)
  {
    long catchUp = Math.min(this.backlog, delay);
    this.backlog = this.backlog - catchUp;
    long sleepTime = delay - catchUp;
    Thread.sleep(sleepTime);
    return delay;
  } else
  {
    this.backlog = this.backlog + (-1 * delay);
    return delay;
  }
}
```

The game time is the in-game time when the RealTimer is initialized. The RealTimer updates this each step with the size of the time step. The time step sets the time each step should take. When the nextStep method is called the begin time is given. This is subtracted from the current time. The result is the used time that it took to finish all steps of the synchronization process as described above. If this is more then the time step size of the lost time (used time - time step), it is added to the backlog. The timing service can immediately continue to the next time step. If the used time is less then the time step, the RealTimer checks its backlog and makes up as much time as possible not exceeding the time allowed in the step (the time step size). This way the backlog is decreased each time there is time left in a step. If the backlog is zero or less then the time left in the step, the RealTimer calls the Thread.sleep() method until the correct time is reached.

The final value given when initializing the RealTimer is a time factor. This can be used to speed up or slow down the game time. This factor increases or decreases the time allowed in each time step, without changing the step time itself and thus to game time update. Using the time factor makes the RealTimer faster or slower then wall clock time(real time).

5.3 Summary

This chapter describes the design and implementation of a prototype service oriented architecture. The architecture is designed specifically for simulation gaming suites. The name of the architecture is Service Oriented Gaming and Simulation(SOGS). The chapter first handles the design decisions for the architecture by looking at the approaches of chapter 2 and 4. The SOGS architecture is designed as a peer-to-peer system using a service broker for service discovery. Message transmission is done using a publish-subscribe system, where the messages are adapted to an intermediate model. Timing and Synchronization is handled by a separate service.

Next the implementation of the architecture is described. SOGS is an architecture written in Java. It uses a library called DSOL for the messaging. DSOL in turn uses Java Remote Message Invocation. The implementation section describes the details of the Java classes that make up the architecture.

Chapter 6

Existing Architectures for Distributed Environments

In the previous chapter we talked about service oriented architectures as a possible approach for a distributed environment. In this chapter we take a look at systems that are currently in use in this field. The research questions associated with this are:

- **RQ3.1** Are there architectures currently in use for simulation gaming suites?
- **RQ3.2** What are the architectural approaches of these architectures?

The goal is to give some background information on other architectures in the field of distributed simulation. In the chapter 7 we will use these to see how well a SOA approach compares to these architectures. The two architectures discussed in this chapter are also part of the RGS work package 1 research, which was performed at the same time as this thesis project [22].

First we look at an architecture that has been developed by the United States military as an architecture for their simulation environments (section 6.1). The architecture is called High Level Architecture (HLA). It is a system that is specifically designed for distributed simulation environments. Research has been done into the ideas of HLA by the department of systems engineering at the faculty of Technology, Policy and Management. They constructed a light weight version based on HLA, called FAMAS, which is described in section 6.2. Finally we give a short overview of multi-player entertainment gaming approaches in section 6.3. Most entertainment games provide a multi-player aspect nowadays, this makes that there is a lot of experience with multi-player gaming in the entertainment games industry. This section is meant to give some extra information on distributed gaming environments.

6.1 High Level Architecture

The Department of Defence of the United States has been researching and working on distributed simulation since the eighties. The Defense Advanced Research Projects Agency (DARPA) started a project called SIMNET (SIMulator NETworking) in the early 1980s. The goal was to create a prototype for a real-time distributed simulator for combat simulation. The project was followed by DIS (Distributed Interactive

Simulation)[14]. DIS was based on the design principles specific to simulation; autonomy of simulation nodes, simulation time constraints, transmission of 'ground truth' information, transmission of state change information only and Dead Reckoning Algorithms. The last three principles are concerned with synchronization of location and positioning of nodes in a game world. More on this in section 6.3 on multi-player gaming.

After DIS came the current standard: High Level Architecture (HLA). HLA builds further upon DIS and SIMNET. It aims at building simulation systems from components. In HLA the collection of all applications is called a federation, the applications themselves are called federates. There is a software component that facilitates the connection between the federates, called Run-Time Infrastructure(RTI) (figure 6.1). The HLA is defined by three components [17]:

- Object Model Template (OMT) provides a common method for recording information and establishes the format of key models (Federation Object Model(FOM), Simulation Object Model(SOM) and Management Object Model(MOM)). Section 6.1.1.

- Federation Rules ensures proper interaction of simulations in federation and describes the simulation and federate responsibilities. Section 6.1.2.

- HLA Interface Specification defines Run-Time Infrastructure (RTI) services and identifies callback functions that each federate must provide. Section 6.1.3.

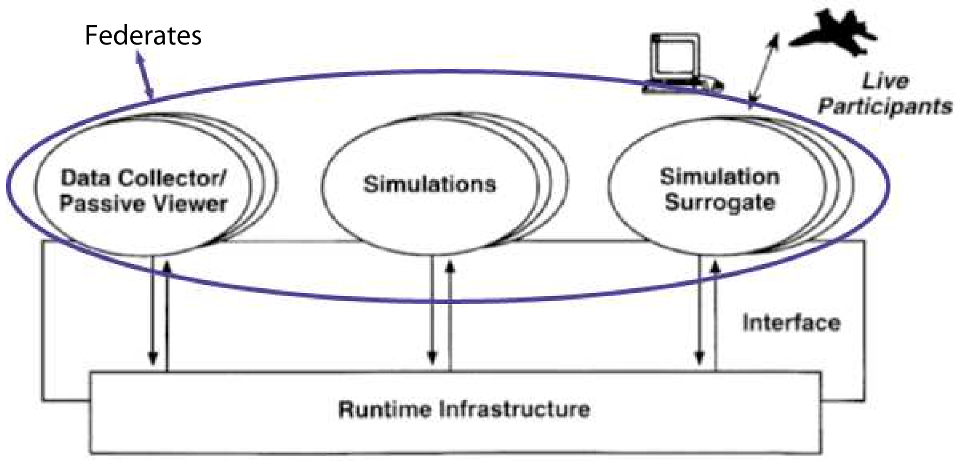


Figure 6.1: HLA federation, with some example federates

6.1.1 Object Model Template

In chapter 4 we have shown the interoperability approaches for communication between applications. HLA uses the intermediate language approach we discussed. 'The HLA Object Model Template prescribes the format and syntax for recording the information in HLA object models, to include objects, attributes, interactions, and parameters, but it does not define the specific data (e.g., vehicles, unit types) that will appear in the object model.' [2]. HLA object models are used to describe a Simulation Object Model, for a federate, or a Federation Object Model for all federates (federation).

'The primary purpose of an HLA FOM is to provide a specification for data exchange among federates in a common, standardized format. The content of this data includes an enumeration of all object and interaction classes pertinent to the federation, along with a specification of the attributes or parameters that characterize these classes.' [2]. Thus describing the intermediate language. An example object of the FOM can be a *Train* object, with an attribute *Speed*.

'An HLA SOM is a specification of the intrinsic capabilities that an individual simulation could provide to HLA federations. The standard format in which SOMs are expressed facilitates determination of the suitability of simulation systems for participation in a federation.' [2].

6.1.2 HLA Rules

There are ten rules that HLA federations and federates must follow. The federation rules are as follows:

1. Federations shall have an HLA FOM, documented in accordance with the HLA OMT.

2. In a federation, all simulation-associated object instance representation shall be in the federates, not in the RTI.

3. During a federation execution, all exchange of FOM data among joined federates shall occur via the RTI.

4. During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification.

5. During a federation execution, an instance attribute shall be owned by at most one joined federate at any given time.

The rules for federates are:

6. Federates shall have an HLA SOM, documented in accordance with the HLA OMT.

7. Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOMs.

8. Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.

9. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of instance attributes, as specified in their SOMs.

10. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation. See section 6.1.3.

6.1.3 The Run Time Infrastructure

The Run-Time Infrastructure(RTI) is responsible for connecting the federates together, in fact it is what makes separate federates a federation. The implementation of the RTI is not set, so various implementations are possible. The restriction is that a federation can only have one RTI. All communications between the federates goes through the RTI. In terms of network deployment this is a client-server architecture (section 4.1), where the RTI is the server and the federates the clients. The RTI provides several services for the federates they will be discussed in more detail below.

- Federation management
- Declaration management
- Object management
- Ownership management
- Time management
- Data distribution management

Federation Management

The Federation management part of the RTI is responsible for creation, dynamic control, modification, and deletion of a federation execution. First the federation execution must be created, then the federates can join and resign from it at will. [1][10].

Declaration Management

All federates have to declare their intent for *information generation* to the declaration management. Declarations must be conform the Federation Object Model of the federation they join. High Level Architecture uses a publish-subscribe mechanism as described in chapter 4. All information a federate declares to share (publish) is available to all other federates. Federates can use declaration management services exclusively, data distribution management services exclusively, or both declaration management and data distribution management services to declare its intention to *receive information*. [1][10].

Object Management

Object Management is responsible for registration, modification, and deletion of object instances and the sending and receipt of interactions. Each object a federate wants to instantiate must get an ID from the object management part of the RTI services. This object ID is then used whenever the federate creates or deletes an object, or whenever attribute updates or interactions are sent. Delivery of the published objects is handled by the object management service. There are two transport services: best effort and reliable. [1][10]. The HLA implementation we used for the tests (PitchRTI, see below) uses TCP/IP protocol and shared memory for the reliable transport service. For the best effort transport service it uses UDP/IP unicast, UDP/IP multicast and shared memory.

Ownership Management

Ownership Management allows responsibility for an object to be shared or transferred between federates. The federate that instantiates an object has ownership over it. It is the only federate able to delete the object. Changing the ownership of an object is handled by the ownership management of the RTI. When an object is deleted all federates are informed to not publish anything related to the object any more. [1][10].

Time Management

Time management is concerned with the mechanisms for controlling the advancement of each federate along the federation time axis. There is one central time axis, which is maintained by the RTI Time Management services. Time advances shall be coordinated with object management services so that information is delivered to federates in a causally correct and ordered fashion. Time advances can be constrained by other federates or unconstrained. At the highest level, the federation appears to the RTI as a collection of federates that communicate by exchanging time-stamped events. Ordering can be done using three different techniques; guaranteed time stamp order, best effort time stamp order and receive order. All messages of the federates are sent to the RTI as time-stamped events. The RTI takes care of the delivery at the appropriate time. The local time of the federates is synchronized with the time axis of the RTI. This mechanism works as described in the *central synchronization mechanism* in section 2.1.2. [1][10].

Data Distribution Management

Data distribution management (DDM) services may be used by federates to reduce both the transmission and the reception of irrelevant data. Whereas *declaration management* services provide information on data relevance at the class attribute level, data distribution management services add the capability to further refine the data requirements at the instance attribute level. It controls publisher-subscriber relationship between federates in terms of object instances and abstract routing spaces. [1][10].

6.1.4 HLA implementations

HLA has become a well-known distributed simulation architecture. Several commercial and non-commercial implementation are in use at the moment, like PitchRTI[1], CERTI[2] and poRTIco[3]. It has been defined under IEEE Standard 1516. For the RGS project it was decided to test a RTI developed by Pitch Technologies, called PitchRTI, which is compliant to the IEEE 1516 standard. It uses the

6.2 Lightweight architecture FAMAS

Distributed simulation architectures have been a research subject within the department of Systems Engineering of the faculty of Technology, Policy and Management at

[1] *http : //www.pitch.se/*
[2] *http : //savannah.nongnu.org/projects/certi/*
[3] *http : //www.porticoproject.org/index.php?title = Main$_p$age*

the TU Delft for some years. In 2005 the FAMAS Simulation Backbone Architecture was introduced in the PhD thesis report of C. Boer [9].

The aim of the FAMAS Simulation Backbone is to provide a flexible architecture for the interoperability among various distributed simulation models. This is done by the component based, modular design of FAMAS. The components can be characterized as technical or functional. The simulation models are considered functional components, while the technical components are well-specified and more constant components that provide common tasks used by the functional components. Figure 6.2 shows how the components are connected to each other. FAMAS uses a publish-subscribe message transmission approach. There are three main technical components

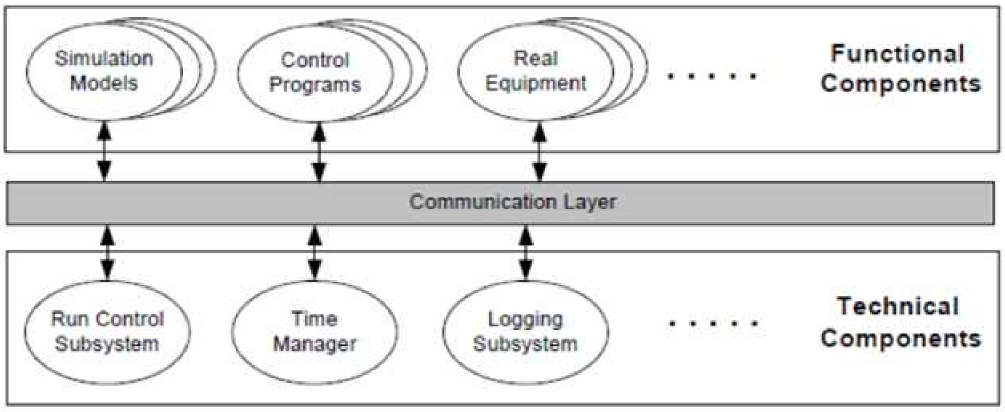


Figure 6.2: FAMAS Simulation Backbone Architecture overview

in the FAMAS architecture:

- Run Control Subsystem: starts, stops and periodically monitors the simulation process.
- Time Manager: synchronizes the simulation time among different simulation subsystems.
- Logging Subsystem: collects logging information from the distributed functional and technical components into a central database.

The same components are shown in figure 6.3. Here we also see another component, the Scenario Object. The Scenario Object completely defines a simulation run of the distributed model. The Run Control subsystem is responsible for the interpretation and execution of a scenario object.

In HLA Terminology, the overall system that consists of technical and functional components is called a federation, where the components that are connected to the backbone are federates. The separately defined technical and functional components give a modular structure to the architecture and allows it to be extended easily with new technical components that allow additional services. The technical and functional components communicate by means of messages. The communication protocol of the simulation backbone that supports this message exchange is Transmission Control Protocol/Internet Protocol (TCP/IP)[22].

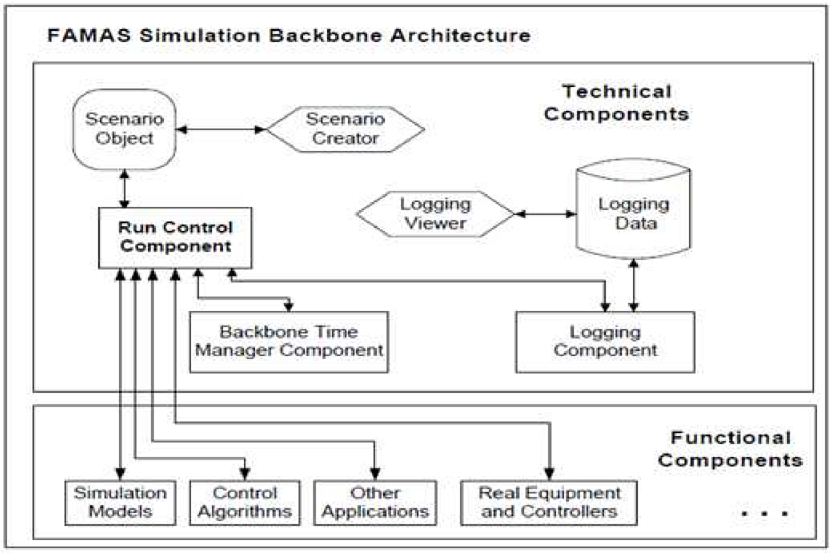


Figure 6.3: A different overview of the FAMAS Simulation Backbone Architecture

In the next sections two of the main technical components will be handled, Run Control and Time Manager. The logging subsystem is not essential for the use of FAMAS. Therefore it will not be discussed further. The connection with the characteristics of distributed architectures (chapter 4) and gaming simulation suites (chapter 2) will be discussed in the sections below.

6.2.1 Run Control

The Run Control subsystem is one of the core elements of the FAMAS Backbone Simulation architecture. It has the control over all the subsystems. The Run Control Subsystem and its interface is coded in Java. The Run Control subsystem is responsible for three main activities[9]:

1. Initialization and start of a distributed simulation execution (see figure 6.4)

2. Special Activities during distributed simulation execution

3. Termination of distributed simulation execution (see figure 6.5)

All technical and functional components are connected to the Run Control. After connection a component can be reached by all other participants, because Run Control knows, registers and makes available its IP-address and port number. All connecting components are known in advance and stated in the scenario object. When a component leaves it must inform the Run Control. The Run Control will in turn inform the other components. The Run Control makes regular checks to see whether components are still connected. Communication between components is done in a peer-to-peer manner, as described in section 4.1.

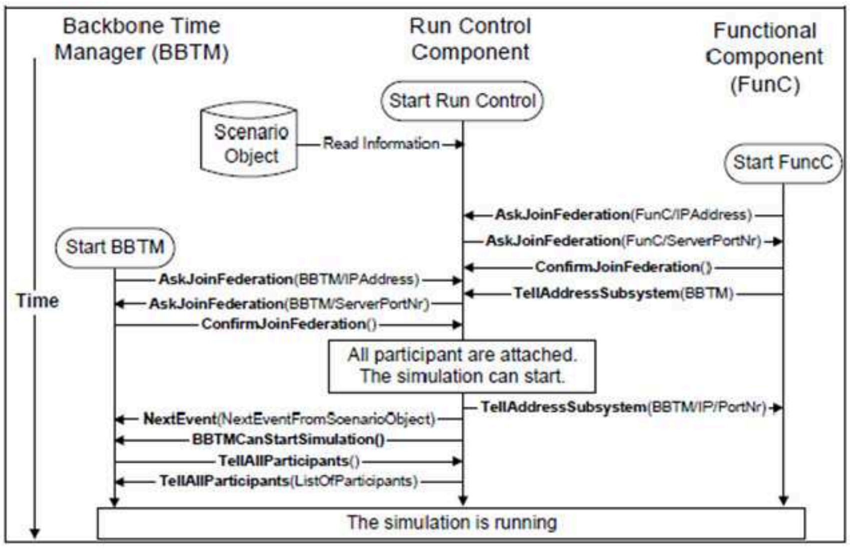

Backbone Time Manager (BBTM)
Run Control Component
Functional Component (FunC)
Start Run Control
Scenario Object
Read Information
Start FuncC
AskJoinFederation(FunC/IPAddress)
AskJoinFederation(FunC/ServerPortNr)
ConfirmJoinFederation()
TellAddressSubsystem(BBTM)
Start BBTM
AskJoinFederation(BBTM/IPAddress)
AskJoinFederation(BBTM/ServerPortNr)
ConfirmJoinFederation()
Time
All participant are attached. The simulation can start.
TellAddressSubsystem(BBTM/IP/PortNr)
NextEvent(NextEventFromScenarioObject)
BBTMCanStartSimulation()
TellAllParticipants()
TellAllParticipants(ListOfParticipants)
The simulation is running

Figure 6.4: Start protocol of FAMAS

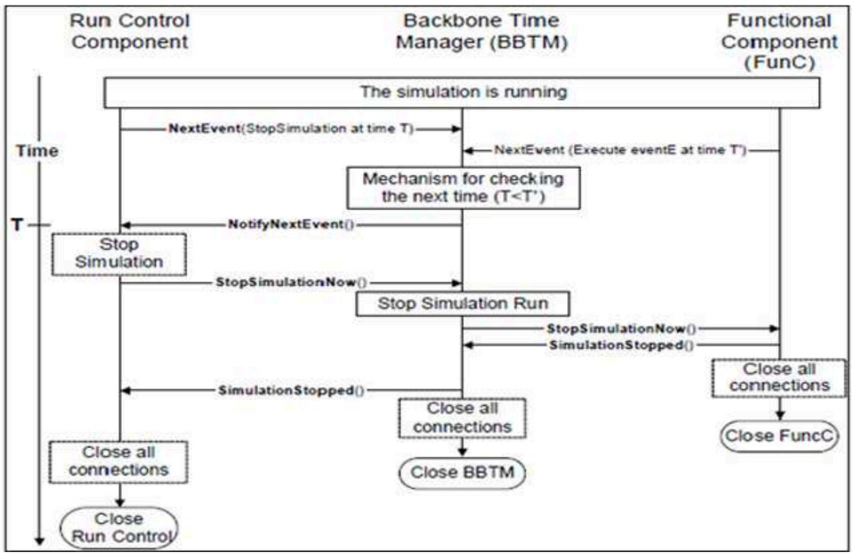


Figure 6.5: Termination protocol of FAMAS

6.2.2 Time Manager

The Time Manager, also called the Backbone Time Manager, is responsible for the synchronization of time of the system. ‘It implements two types of time synchronization mechanisms, namely conservative and real-time. Conservative time synchronization is desired in order to achieve synchronization between discrete-event simulation models, while real-time synchronization aims to provide support for experiments when real equipment is involved’[9].

‘The basic principle for synchronizing the activities of the participants on the same time axis using a conservative mechanism is as follows. Each participant is assumed to send its first future event time, as a next event time stamp, to the backbone time manager as a NextEvent message. Then the time manager selects the participant with the smallest time stamp event and gives permission to perform this event by sending a NotifyNextEvent message. After completing the event, the participant sends its next future event time stamp to BBTM again. Participants sending the same event time are handled in first in first out (FIFO) sequence: the one who sent its event time first is allowed to proceed first.'[9] The real-time mechanism uses the same mechanism, but then the next event steps are restricted by time. The simulation time is subdivided in a sequence of equal sized time steps, and the simulation advances from one time step to the next. This mechanism is similar to the *distributed serial synchronization mechanism* discussed in section 2.1.2.

6.3 Multi-player entertainment gaming

Multi-player entertainment gaming is a very wide concept. There are many different ways it is approached in the entertainment gaming industry. In this section we will provide a short overview of the technologies used. This section serves the purpose of providing additional information on distributed environments in gaming. The techniques described in this section will not be specifically handled as an approach for a gaming simulation suite.

According to Smed et al.[27] there are three distinct classes of distributed interactive real-time applications; military simulations, network virtual environments(VEs) and multi-player computer games. While VEs simulate (possibly real-world) environments, computer games do not necessarily belong to simulations or VEs[27]. The different classes overlap a little, as can be seen in figure 6.6.

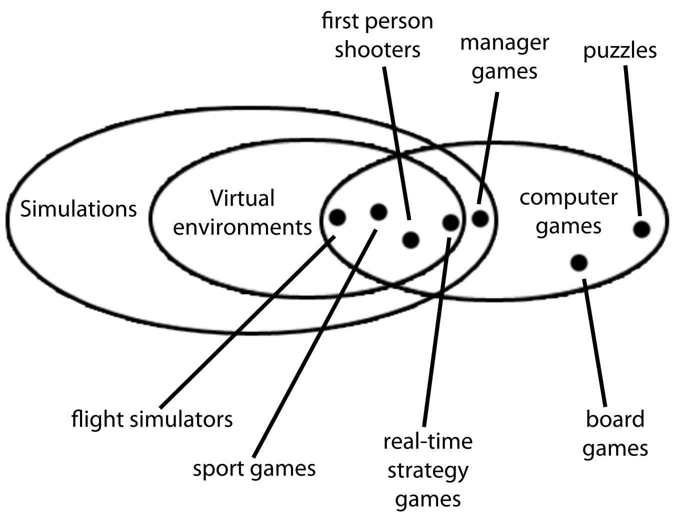


Figure 6.6: Relationship of simulations, virtual environments (VEs) and computer games.[27]

For multi-player the same distributed architectural approaches are available as shown in section 4.1. One of the main concerns has been to get a high performance on the network. Independent from the architectural approaches there is much to gain from reducing the bandwidth requirements. The most common techniques are; packet compression and aggregation, interest management and dead reckoning [27].

Message Compression and Aggregation

There are two main techniques to reduce bandwidth requirements when we look at the messages that are sent between applications, namely compression and aggregation. With compression we try to reduce the size of single messages. This can be done by either lossless techniques or lossy techniques. With a lossless technique all data is preserved, but there is a limit to the size reduction, up to approximately half the size of the original message[27]. Lossy technique are able to reduce the size further, but at the cost of some of the data. Using either of the techniques is dependent on the requirements of the applications.

Message aggregation merges several messages to reduce overhead, like headers. Depending on the size of the actual data the reduction of total data send across the network can be significant. For example two messages with data size 24 bytes and a header of 24 bytes (total 96 bytes), compared to a merged message of 48 bytes with a header of 24 bytes (total 72 bytes). Two techniques for this are timeout-based and quorum-based [27]. In the timeout-based approach all messages before a certain time limit are merged and sent. In a quorum-based approach a predefined number of messages are merged. A combination of these techniques can be used as well.

Interest Management

Interest Management is a technique to filter data. Most of the time the entities in a system are not interested in the entire system. By specifying what information is of interest for them the amount of data sent across the network can be greatly reduced. This interest in data is often called an aura or area of interest[6]. Interest management with auras is always symmetric. So when two auras intersect they are aware of each other.

To achieve a finer-grade message filtering the aura is subdivided in a focus and a nimbus, being the observer's perception and observed objects perceptivity respectively [6]. Now the focus of one entity needs to intersect with the nimbus of the other entity to be aware of it. An example is given in figure 6.7. In the figure we see a game of hide and seek. The seeker is not aware of the hider, since its focus does not intersect the nimbus of the hider. The hider is aware of the seeker, since the hider's focus intersects the seeker's nimbus. The filter based on the aura technique we have shown here is called an intrinsic filter. Another possibility is an extrinsic filter, where the data delivery is based on network attributes (e.g. address). Extrinsic filters are faster to process than intrinsic filters, and even the network itself can provide them [27].

A way to implement these techniques is by using multi-casting. For example the publish-subscribe mechanism described in section 4.2.2 supports this kind of technique.

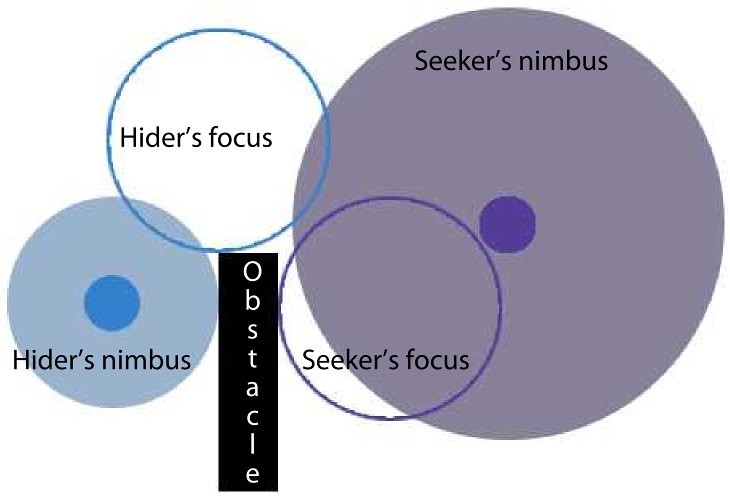


Figure 6.7: Game of hide-and-seek.

Dead Reckoning

Another approach to decrease the number of packages on the network is dead reckoning [26]. Dead reckoning is used to synchronize locations of objects between distributed applications, for example player movement in a distributed multi-player game. This technique uses approximation to provide the loss of information. The approximation predicts movement of objects based on previously received information. The prediction is done based on velocity information. Since it is an approximation the real location can be different from the calculated location. By sending information on the exact location and a convergence technique the objects location is synchronized with the location in the source application. Without the convergence technique the node would jump to correct location every time an exact location update arrives. This would result in jerky movement of the nodes in the visual environments.

Without the dead reckoning technique exact location updates must be send regularly. The technique decreases the time and update needs to be send and solves this with the approximation in the receiving node. There are different implementations where a trade-off is made between the amount of messages send and the calculation made at the receiving node. Different implementations can be used in the same system for different object types.

6.4 Summary

In this chapter we have discussed three approaches for connecting games and simulators in a distributed environment. The goal is to provide background information on the architectures used to compare to a service oriented approach. This supports the sub question: 'How well does a SOA approach compare to systems currently in use?' We started with a popular system in distributed simulation, called High Level Architecture. This is an architecture specifically design for distributed simulation environments by the United States Department of Defense. Next a architecture based on HLA is discussed. It is called FAMAS and has been developed as a PhD project within the department of Systems Engineering of the faculty of Technology, Policy and Management at the TU Delft. The HLA and FAMAS architectures are part of the comparison to evaluate SOA as a distributed gaming simulation approach.

Finally we discussed some techniques commonly used in multi-player entertainment games. These provide some additional background information on distributed gaming systems, but are not part of the evaluation of SOA.

Chapter 7

Evaluation

Up to now we have looked at what a gaming simulation suite is and based on an example suite we have determined the important requirements of such a gaming simulation suite. We have described several different ways to construct an architecture to facilitate the connection between the applications in the suite. In this chapter we look at how a service oriented architecture can support the quality attributes we determined for the example gaming simulation suite. This chapter is mainly concerned with the research questions:

> **RQ2** How well does a service oriented architecture support the requirements of gaming simulation suites?

and

> **RQ3.3** How well do the other architectures support the requirements of gaming simulation suites?

In the previous chapters we have looked at three architectures, SOA, HLA and FAMAS. The evaluation is performed on all three of them. For the service oriented architecture we will look at the principles of SOA and we have constructed a prototype SOA (SOGS) to do some tests with. For the evaluation of HLA we have looked at one specific implementation, namely Pitch RTI. The FAMAS architecture has only one implementation and thus will be evaluated as it is. Of all three architectures we have made a test implementation and ran some experiments with them.

The evaluation of the architectures has been a team effort of the Railway Gaming Suite research team. The Pitch RTI and FAMAS implementations were research subjects of other members of the team. The results in this evaluation are mainly based on their efforts. The SOA research and implementation of SOGS are the focus of this thesis. The evaluation of a SOA solution is therefore based on the implementation as carried out in the context of this research project.

First we will take a look at the methods that are used for the evaluation. Then we will go into the experiment set-up we used as part of the evaluation. After which we will look at the quality attributes from chapter 3 again. In the subsequent sections the quality attributes are evaluated for the three architectures. After the evaluation we discuss how SOA compares against the others in the next chapter. Here we will also look at the sensitivity points we found during the ATAM. This will give some insight into how the requirements influence each other.

7.1 Evaluation methods

The evaluations of the architectures will be done using three methods: implementation details, hand-on experience and tests.

- **Implementation details**: In chapters 4, 5 and 6 we discussed the implementation details of the three different architectures. The descriptions of these chapters are used to see if the architectures support the requirements.
- **Hand-on experience**: By making an implementation for the three architectures we got some hand-on experience on the use of them. The SOGS architecture was used for the SOA part. The implementations of the other two architectures (HLA and FAMAS) were made by two other members of the RGS team and they shared there experiences for this thesis report. This hands-on experience helped with the getting a feel of how easy it is to construct a simple system with the architectures.
- **Tests**: For the real-time performance requirement a test set-up was implemented for all the architectures. The specifics of the test set-up are described in section 7.2. The results of the tests showed us how well each architecture performed with respect to this requirement.

In this evaluation we use the same value systems as is used in the ProRail WP1 report [22]. For each requirement the architectures get assigned a value of -1,0 or +1 to represent that the implementation of the requirement is a risk, borderline, or a non-risk, respectively for the given architecture.

7.2 Performing the Test

For the evaluation of some of the requirements we devised some tests. The test are based primarily on the performance requirements. The test are implemented by means of an experiment set-up. The main thought behind the tests is: How many messages can be send between applications in a specific time frame? From the number of messages we can then derive whether it is enough to support the performance requirement. By implementing the experiment set-up and performing the tests we get insight into using the different architectures which helps with the evaluation of the other requirements.

In total we performed 10 experiments for each architecture, with different application set-ups and different payload. These variables are further explained below.

7.2.1 The Experiment

The goal of the experiment was to test how many messages we could send from one application to another within a specific time frame. We do this by finding out how much time it takes for the architectures to get a message from an application, transform it and send it to a receiver, transform it back and give to the receiving application. The same message is transformed again at the receiver side and send to the original sender, where it is transformed again and given back to the first application. In other words the round trip time of data published by an application. In order to keep this timing as accurate as possible we wanted control over the number of messages at a time on the network. Therefore we created two applications that are totally dependable on each other for sending messages. So on one side a message is sent, it then waits for the return message before sending a new message. The other side waits to receive a message and sends it back. By controlling the number of sender-receiver pairs we can control the number of messages on the network.

Since all three architectures were written in Java we constructed two Java applications for the roles of sender and receiver. Each of the experiments were performed with the same sender and returner applications. A schematic overview of the experiment set-up is shown in figure 7.1. The applications work as follows:

- The sender application starts sending a message as soon as a session starts. Waits until the sent message is returned and then sends a new message. The send and receive/returned times of each message are logged. At the end of the session return times of each message are calculated as well as the number of messages returned in a specific time frame. The sender can change the payload of the message. See below for more information on the payload sizes.

- The returner application waits to receive a message. As soon as a message is received it immediately sends it back to the sender.

Senders and returners are coupled in a one-to-one relation in these experiments.

One of the variables we want to test is the payload of the messages, see below in section 7.2.2. We want to know the exact size of the message payload. To achieve this we used the following method. We chose to use a Java String consisting of a 7 digit identifier followed by 40 ‘1s’. With the length of a Java String we could precisely make messages of a specific payload size. In Java the empty String takes 40 bytes, for the first three characters this does not change, then for every four characters it increases by 8 bytes. So a String of length 0-3 has size 40 bytes, 4-7 has size 48, 8-11 has size 56 bytes, etc. The sender creates a default payload of 128 bytes by making the String of 47 characters. At the start of an experiment it gets a multiplier to increase the payload. This works according to the function: $128 * 2^{multiplierbytes}$, for example a multiplier of 3: $128 * 2^3$ = 1024 bytes = 1 kilobyte (kb).

Each experiment run lasted 11,5 minutes. During a run the results were logged 23 times after 30 seconds each time. Each time frame of 30 seconds we call a replication. The first two and last replication were neglected. They served to give the architecture time to start up and finish. During the test with multiple sender-returner pairs (see below) these were not started simultaneously. Since the system is distributed the start signal does not arrive at each application at the same time. The synchronization mechanism however made sure this was solved after the first message. This made that the first round trip times were not average measurements. In the results we saw this was indeed the case. Furthermore the replication times were handled locally by the sender applications, this means they we not synchronized exactly. Therefore during the last replication not all application finished at the same time, influencing the performance.

The idea to neglect the first two and last came from a Java Message Service[1] performance test done by Krissoft[29]. Their reason to do this was: ‘The first two and

[1] $http://www.oracle.com/technetwork/java/index-jsp-142945.html$

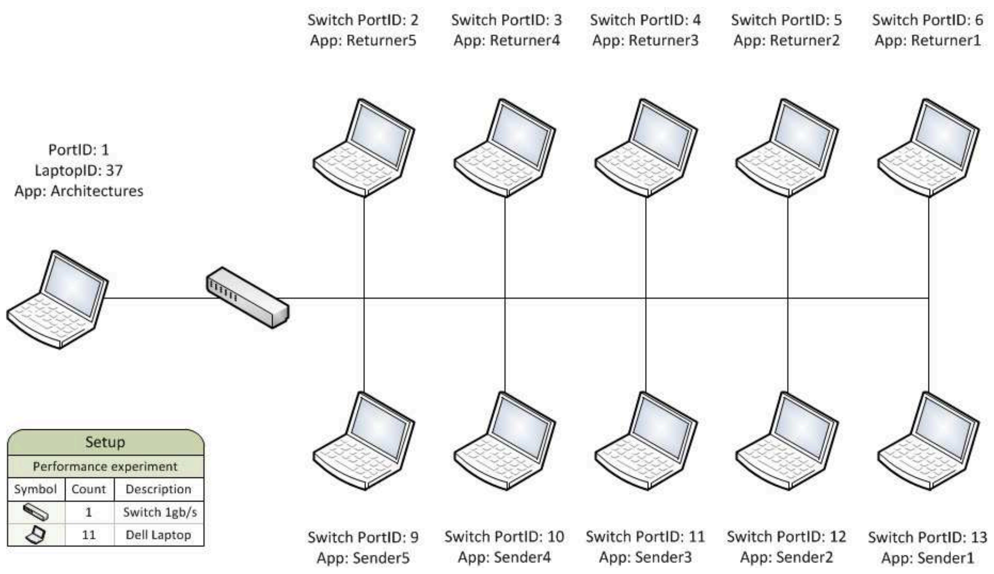


Figure 7.1: Schematic set-up of the experiment

last intervals were considered ramp-up and ramp-down intervals, respectively. Ramp-up intervals are times during which the systems are increasing their message handling capacities, typically via resource allocation, in response to the newly introduced client load. Similarly, during ramp-down intervals, the systems are decreasing their capacity in response to decreased client loads that result from test completion.' For us this seemed a good precaution to take together with the reasons specified above. Therefore we did the same. From the results it appeared neglecting the first and last would have been sufficient.

We have extracted some graphs from the experiments showing the effects of increasing payload and the effect of having more applications communicating in the same architecture environment (two versus ten). These graphs are shown during the evaluation in section 7.4.

One of the requirements for the gaming and simulation aspect of the suite is to have a synchronized system. Therefore the implementations for the experiment need to conform to this requirement. As shown in the description of the architectures (chapters 4 through 6) each of them supports a synchronization method. The experiments were implemented with these methods activated.

7.2.2 Experiment variables

As stated above the experiments were performed ten times for each architecture. We used two variables to get the results we wanted. The first variable is the number of applications in the system, the second is the payload size of the messages sent across the network.

We vary the number of applications between two and ten, excluding the application used by the architecture. A minimum of two is required because we need a sender and a receiver for the experiment. Ten is the maximum for this experiment because of licence agreements for the Pitch architecture, allowing for a maximum of ten federates.

For the payload we looked at the current implementation of applications of the Railway Gaming Suite. Currently there is direct communication between the FRISO simulator and the traffic controller game PRL. We have made a log of the messages sent between the two applications. Currently they use a XML format for the data that is sent between them. From the log we could conclude that most of the messages are small, less then 200 characters including XML tags. We also saw that some of the messages were bigger, 6.919 and even bigger yet, 727.216 characters including XML tags. These numbers correspond with payload sized of around 0.5 kilobytes, 14 kilobytes and 1500 kilobytes. With these numbers in mind we decided to use the following multipliers for the sender application; 3, 6, 9, 12 and 15 corresponding with payload sizes; 1kb, 8kb, 64kb, 512kb and 4069kb, respectively.

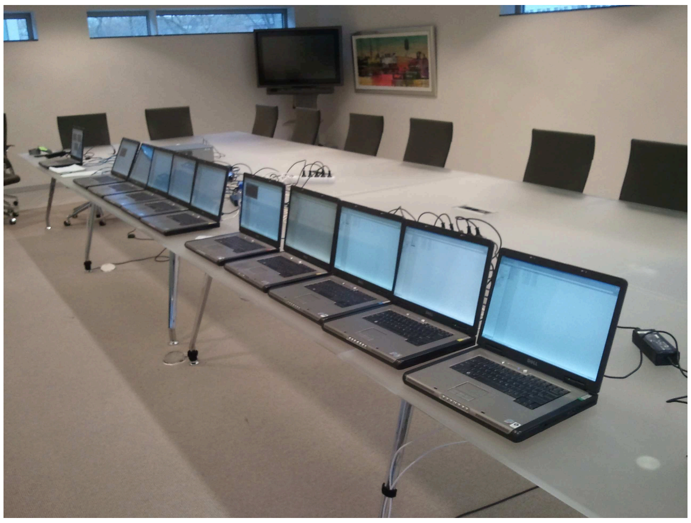


Figure 7.2: The set up of the experiment with the eleven laptops.

Figure 7.3: Close up of one of the laptops used for the experiments.

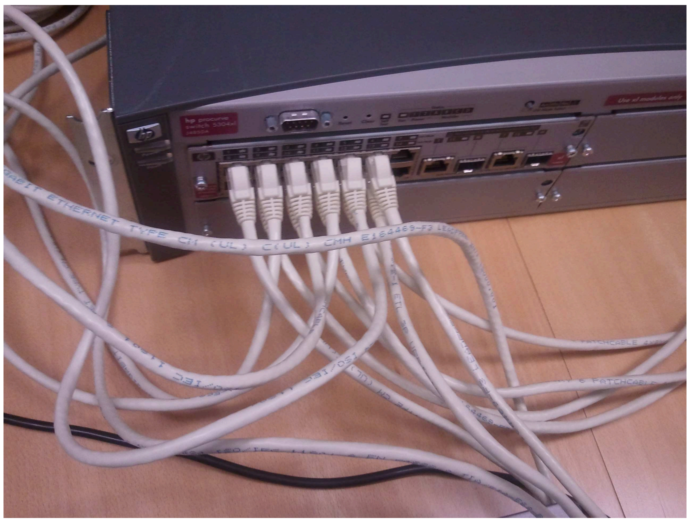


Figure 7.4: Close up of the switch used for the experiments.

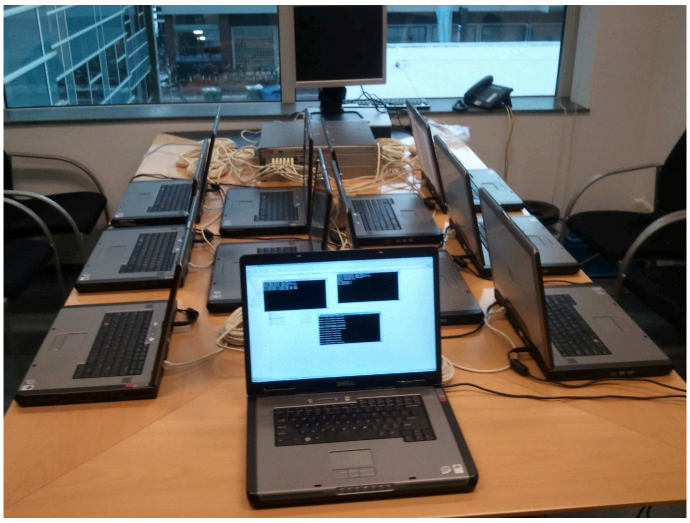


Figure 7.5: The set up of the experiment during another session.

7.2.3 System environment

In order to make the experiment results as valid as possible we have taken some measures in order to avoid influences from outside the experiment environment:

- All applications in the test set-up ran on a separate laptop.
- Architectural software, Pitch RTI, ServiceBroker and SynchronizationService (SOGS), and RunControl and BackBoneTimeManager (FAMAS) ran on the same separate laptop.
- The laptops used in the experiments were identical high-end Dell XPS 17” notebook computers that TU Delft uses for simulation (SimLab) and gaming (Game-

Lab) purposes. Only difference was the size of the hard drive, which was 120Gb on some and 160Gb on others.

- An industrial grade, 1 Gbps switch was used to run the network. This switch was the same one for each experiment.
- Newly bought Cat6 cables were used for the network. The same cables were used in all experiments.
- Network traffic was monitored in order to find and disable unnecessary use of the network.
- For each experiment, the same image of Windows XP service pack 3 with several simulation environments was loaded onto the laptops. Therefore, no effects from prior experiments or prior use remained on the laptops.
- Windows update functions and other disturbing factors such as indexing services that could cause a load on the computers during the experiments were switched off as much as possible. The computers were given ample time for initial tasks to be carried out (some software tests for updates when the operating system starts).

Specifics of the laptop settings can be found in Appendix C. The physical set-up is show in the pictures 7.2 through 7.5.

7.3 Quality attributes

The quality attribute requirements we acquired during the ATAM (see chapter 3) serve as evaluation points for the three different architectures. Below we show the requirements again and specify which evaluation method we will use.

Quality Attribute	Specific Requirement	Evaluation method
Performance	(P1) Real-time play	Tests
	(P2) Redundancy	Implementation details
Extendibility	(E1) New applications	Implementation details & Hands-on experience
	(E2) New simulation components	Implementation details
	(E3) New ontology	Implementation details & Hands-on experience
Consistency	(C1) Time paradigms	Implementation details
	(C2) Causality	Implementation details
	(C3) Synchronization	Implementation details
Availability	(A1) Session Duration	Tests & Implementation details
	(A2) Failure Detection	Implementation details
Flexibility	(F1) Support new players	Implementation details
Usability	(U1) From session-to-session	Implementation details
	(U2) Scenario set-up	Implementation details

Maintainability	(M1) Documentation	Implementation details & Hands-on experience

Table 7.1: Evaluation methods for the quality attributes

7.4 Performance

There are two requirements we specified for the performance attribute. The two requirements are; real-time play and redundancy. In table 7.1 it is shown that we evaluate the first requirement using tests and second based on implementation details.

7.4.1 P1 Real-time play

We start with real-time play requirement. This requirement requires the architecture to handle up to 50 applications in real-time. Real-time in this case means synchronized with a wall clock, as discussed in section 2.1.1. Related to this is that it should be able to meet this requirement while keeping the system synchronized. As discussed in chapters 5 and 6 all architectures include a synchronization mechanism. The synchronization mechanism of each architecture was integrated in the experiment implementation for the tests, see section 7.2.

In order to get some idea of what number of messages is required to keep the system real-time we look at the RGS applications that are already implemented, and multi-player computer games. From the FRISO-PRL log (see section 7.2.2) we extracted that the number of messages sent by one of the applications in one second ranges from less then one up to around 15 at peak times. On a side note, in the current implementation of communication between FRISO and PRL the game runs in real-time. This number of 15 messages per second gives us an indication for only the FRISO-PRL case. We can imagine that when we connect more games and simulation this number increases. Furthermore, some applications might send more information across the network, for example two train driver games sending location updates to each other. In a research study of network game traffic [15] it is shown that in the first person shooter game 'Counter Strike' [2] the number of update messages per second are a little less then 30 for each client. We take this as an upper bound for the games in a simulation suite. In the same report by Farber [15] it is said that 99% of the packages are smaller then 0.25 kilobytes and none larger then 1.5 kilobytes. We take the 30 messages per second as a threshold to evaluate this requirement.

The experiment we described above was designed to test the real-time play requirement. The experiment is designed to calculate the return times of a message. The total dependency of the sender and receiver resulted in the fact that there was only one message (in case of one pair) or five messages (in case of five pairs) on the network at a time. By having this strict message limit to the number of message we can see the exact influence of having five times as many messages on the network. In case we did not implement this strict message limit, we would not know how many messages

[2] $http://en.wikipedia.org/wiki/Counter-Strike$

were on the network at the same time. We expected other influences could play a part without the limited number of messages, like the hardware we used.

The one pair case shows the unhindered return times and the five pair variant shows the effect of having more messages on the network at the same time. The figures 7.6 and 7.7 show the results of the experiments. The results are given in return messages per 30 seconds. The fact that we use return times means we need to multiply the number by 2 to get the results of the messages one application can send. To compare it with the real-time requirements we translate this to messages per second (m/s). From the graphs we take the minimum results of the architectures to account for the worst case scenario.

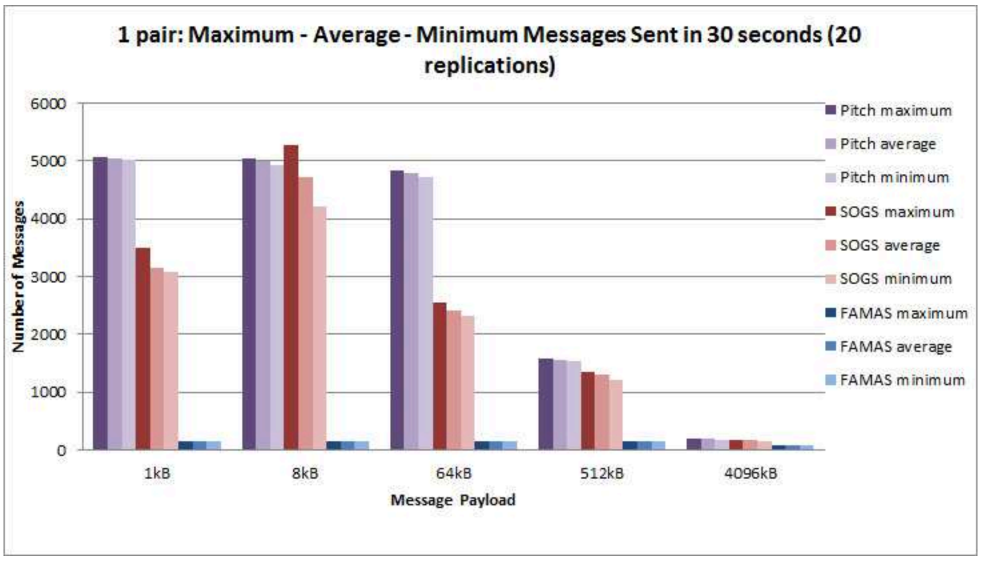


Figure 7.6: Results of the experiments with 1 application pair.

Given the results we can see if the implementation of requirement represents a risk or not for the three architectures.

The SOGS implementation is able to handle more then 3000/30*2=200 m/s with a payload of 8 kilobytes or less in an environment with just 1 application pair. As the payload increases this number drops to a around 150/30*2=10 m/s for the 4096 payload messages. For the case with five pairs we see that SOGS is able to handle 2500/30*2=166 m/s for a payload of 8 kilobytes or less, this drops to around 90/30*2=6 m/s for a payload of 4096 kilobytes. Comparing this to the threshold of 30 m/s the SOGS implementation meets the threshold for small payloads, which make up the majority of the messages. The larger messages might take more time, but the catch-up mechanism of the RealTimer of the SynchronizationService of SOGS can make up for this. We see from the graphs that the number of messages per second drops as the number of applications grows. Given the large overhead on the number of messages we expect that it stays larger than the threshold of 30 m/s. Furthermore the SOGS implementation is only a prototype and with optimizations the performance can be better, without deviating from the SOA principles. Taking all this into account the SOGS implementation of a SOA architecture receives a +1 score.

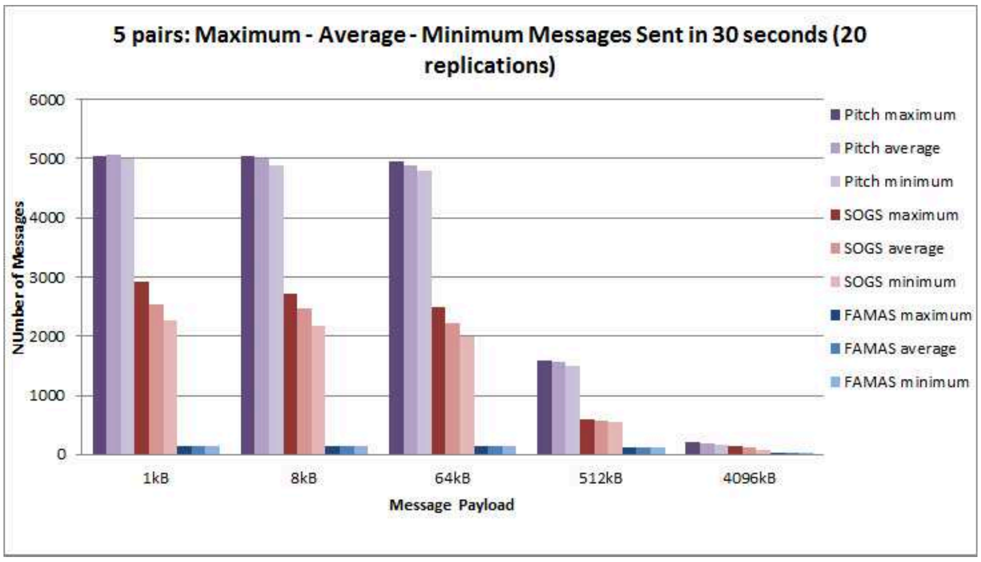


Figure 7.7: Results of the experiments with 5 application pairs.

For the Pitch RTI implementation we see it is able to handle 5000/30*2=333 m/s, for a payload of 8 kilobytes or less in the 1 pair setting. This drops to 180/30*2=12 m/s for the larger messages. In the 5 pair setting we see it is still able to handle 5000/30*2=333 m/s and 150/30*2=10 m/s for the 8 kilobytes or less and up to 4096 kilobytes messages respectively. This is more then enough to meet the threshold of 30 m/s and thus PitchHLA receives a score of +1.

The last architecture implementation is that of FAMAS. We see this implementation never goes over 150/30*2=6 m/s. For the higher payload this drops to 80/30*2=5 m/s and even 15/30*2=1 m/s in the five pair environment. We can directly say this is not enough to meet the threshold and this this posses a risk and FAMAS receives a score of -1.

7.4.2 P2 Redundancy

The redundancy requirements looks at the number of unnecessary messages. To get a +1 score in this respect the architecture must be able to support a message filtering technique. Since all three architecture use a publish-subscribe system for the message transmission they all get a score of +1.

7.4.3 Performance evaluation results

Table 7.2 shows the scores of the evaluation of the performance requirements.

7.5 Extendibility

The extendibility quality attribute is about adding new applications or functionality to an existing architecture. In chapter 3 we refined this attribute into three requirements; new applications, new simulation components and new ontologies. In this section we

	P1	P2
SOA/SOGS	+1	+1
HLA/Pitch RTI	+1	+1
FAMAS	-1	+1

Table 7.2: Scores for the performance evaluation

will look at the how easy it is to extend the architectures. We do this by looking at the implementation details and base it on the hands-on experience we got during the implementation of the experiment applications. In this section we look at the ease of implementation based on the principles behind the architectures and the hand-on experience. In the section on the maintenance quality attribute (section 7.10) we look at the support to help with the implementation of new applications and functionality.

7.5.1 E1 New applications

In an evolving gaming suite it is important to be able to add new games and simulators to an existing system. This requirement looks at the ease of making a new game or simulator part of the system.

One of the characteristics of SOA is that the services are loosely coupled. This means that adding a new service should have almost no consequences for the other services. For the implementation of the experiment using the SOGS architecture it was easy to make the two new services, sender and returner, using the BasicService class. Only the application's specific functionality had to be implemented. The connection of the services is done by the ServiceBroker, which means the programmer does not need to do this manually.Services do not need to know the specifics of other services, only what they have to offer and what they want to have. Score +1 for the SOA principles supporting this requirement and the SOGS implementation proving it.

The federation/federates structure of HLA is designed to allow new application to be connected to the architecture. The RTI takes care of the architectural connection between the federates. Connection of new applications should therefore not take to much effort. During the implementation of the experiment using the Pitch RTI proved to be easy. The experiment was implemented using an example federate and transforming it to connect to the experiment applications. HLA gets a score +1.

The FAMAS Simulation Backbone is designed and developed in a modular way so that that if new federates need to be added or adapted it should not influence the other federates [22]. This means adding new components should not require too much effort. During the implementation of the experiment it appeared it was not so easy to just use an existing FAMAS federate and make a new one with it. Even though FAMAS is designed to support the easy integration of new applications, the actual implementation requires detailed information of the architecture itself. It gets a score 0.

7.5.2 E2 New simulation components

Next to adding new games and simulators to the system it should also be possible to add new functionality to the architecture, like an integrated logging system for example.

For a service oriented architecture the same goes here as for adding new applications. The loosely coupled nature of the SOA paradigm makes it easy to replace or add a service. Looking at the SOGS implementation we can clearly see this. The architectural components are divided in separate services; ServiceBroker, SynchronizationService. These can be changed or new components added. For example a logger service can be added which subscribes to all messages. Based on the SOA characteristic of services being loosely coupled it gets a score: +1.

The core functionalities of HLA are part of the RTI. These functionalities are highly interdependent. In case of a open source version of HLA adding the functionality should be possible but requires extended knowledge of the inner workings of the Run Time Infrastructure. In a commercial version of HLA, like Pitch RTI this part is not accessible. Therefore changing of extending this functionality is not easy or impossible. We conclude this is a risk of using Pitch RTI and HLA, thus it gets a score of -1.

The FAMAS architecture is built up from different components. Its fundamental technical components like the Run Controller and Backbone Time Manager can be extended with the addition of new ones such as loggers. Moreover, FAMAS is extensible in a way that it allows for replaceable elements (e.g., it allows simulation vendors and practitioners to replace already existing functionality with a more efficient one) [22]. Due to the modular design FAMAS gets a score of +1.

7.5.3 E3 New ontology

Extending the ontology of a gaming simulation suite means extending the type of messages that can be send between the applications.

In chapter 4 we have seen that one of the possibilities for message transfer is to use an intermediate language. In the case of a new ontology this means the intermediate language model should be extended to support the new message types and the applications that use these message types need to extend their adapters. The SOGS implementation supports the intermediate language model. The DataModelBuilder Tool (see Appendix B) provides the means the create new message types. This shows that a service oriented architecture (SOGS) is capable of supporting this requirement en it thus gets a +1 score.

In the Pitch RTI and FAMAS architectures, a data dictionary or FOM is available, which makes ontology extension easier and less risky [22]. The FOM acts like the intermediate language. Pitch RTI has a specific tool for easy extension of the FOM. Even though FAMAS has no such tool extending the FOM does not provide a risk for the system. Both architectures get a +1 score.

7.5.4 Extendibility evaluation results

Table 7.3 shows the scores of the evaluation of the extendibility requirements.

	E1	E2	E3
SOA/SOGS	+1	+1	+1
HLA/Pitch RTI	+1	-1	+1
FAMAS	0	+1	+1

Table 7.3: Scores for the extendibility evaluation

7.6 Consistency

Three important aspects of simulation gaming are; timing, synchronization and causality (see chapter 2). For each of these requirements we will look how they are handled in the architectures. During the evaluation we will consider the implementation details of the architectures.

7.6.1 C1 Time paradigms

The ability to handle multiple time paradigms means being able to pause the game, run it in continuous time mode (real-time as well as faster/slower then real-time) and run it in discrete time mode.

Timing of services is specific to gaming and simulation. There is no SOA characteristic that takes care of this. However the SOA paradigm does not prohibit this requirement. The SOGS implementation is able to handle multiple time modes (real-time, slower and faster then real-time). Extending this with a pause function and the ability to run in discrete time mode should be possible. Score +1.

Handling multiple simulation federates with different time paradigms was one of the initial requirements of the HLA framework. As a result, it provides all the necessary services at the architectural level to manage multiple time paradigms within the same model [22]. Score +1.

The FAMAS architecture is a lightweight version similar to HLA and contains the appropriate services to manage multiple timing paradigms [22]. Score +1.

7.6.2 C2 Causality

In a simulation game, it is important that a strict causality is respected between components. This feature is generally fulfilled by a central component which grants time advance authorization to the participating components [22].

The SOA paradigm does not provide specific solutions in this respect. The SOGS implementation however shows that it is possible to integrate it in a system based on the SOA paradigm. In SOGS it is part of the messaging structure. The SynchronizationService in SOGS enforces the causality as a separate service by delivering the messages by time stamp. Score +1.

In HLA the Run Time Infrastructure has a time management component (see section 6.1). Time management is able to deliver events to the federates in order of time stamp. This ensures the strict causality we need in the system. Thus HLA gets a score of +1.

The FAMAS Simulation Backbone has a dedicated component as well. The Backbone Time Manager takes care of the delivery order of events. This way strict causality is maintained in the system. FAMAS gets a score +1 for this requirement.

7.6.3 C3 Synchronization

In the previous sections we discussed the time paradigms and the causality of the system. These are closely related to synchronization of the system. All three architectures have a dedicated service/component for this (here we use SOGS as a SOA). This results in a +1 score for all three architectures.

7.6.4 Consistency evaluation results

Table 7.4 shows the scores of the evaluation of the extendibility requirements.

	C1	C2	C3
SOA/SOGS	+1	+1	+1
HLA/Pitch RTI	+1	+1	+1
FAMAS	+1	+1	+1

Table 7.4: Scores for the consistency evaluation

7.7 Availability

The availability quality attribute is concerned with the architecture being able to keep running during a session of several hours. To support this the system should be able to detect some errors and allow the facilitator to solve them.

7.7.1 A1 Session Duration

Keeping a session running means the architecture should be able to handle failures in the system and monitoring mechanisms to check on all the applications.

The SOA paradigm does not have characteristics specific to handling of failures or monitoring of services. The SOGS prototype architecture has no functionality currently built in, but the UniqueTimedEvent is created with this in mind. Monitoring mechanism can also be easily implemented as part of a service. Each service can have a monitor for example checking every second whether a service is still up, if not send a message to the ServiceBroker. Even though there was no specific failure system in the SOGS architecture the experiments ran without problems. Since implementation of a service oriented architecture is open and the SOA paradigm does not prohibit implementation of fault handling or monitoring mechanisms it gets a +1 score.

The Pitch RTI is a commercial product. It has been tested and used for many projects. Pitch RTI supports fault tolerance of unstable federates, heart-beat and broken link detection and automatic resign of crashed federates. This makes that it is designed to be robust. The experiment implementation of the Pitch RTI showed it could run several times for eleven and a half minutes without problems. It gets a score of +1.

Since the FAMAS architecture is a research product it allows for enough openness and flexibility to implement new mechanisms and functionality. The current implementation does not prohibit the extension of failure handling mechanisms. Considering the possibilities to extend the FAMAS Simulation Backbone it gets a score of +1.

7.7.2 A2 Failure Detection

In a simulation gaming environment errors can occur due to human interaction. This can make that the system stops. In this case it should be possible for the facilitator to influence the system so it can continue running.

The loosely coupled characteristic of SOA makes it ideal for an extra service, like a facilitator service, to exist. This service can then subscribe to relevant messages to monitor the state of the system. A publish-subscribe system like that of SOGS can help with getting the relevant information to such a facilitator service. There should be no risk in implementing this. Score +1.

In Pitch RTI there is a logger that allows users to detect an interruption in the execution of one of the participating federates [22]. Using this logger a facilitator services can be constructed. Score +1.

Concerning FAMAS, logical faults at the architectural level can be easily detected using the logger functional components. FAMAS components, whether functional or technical, are designed to produce certain types of failure messages, messages showing the internal communication between components, a process tracking messages and showing the number of messages between components. All of these mentioned properties allow user to track the failures or anomalies easily [22]. This provides the functionality to implement a facilitator service. Score +1.

7.7.3 Availability evaluation results

Table 7.5 shows the scores of the evaluation of the extendibility requirements.

	A1	A2
SOA/SOGS	+1	+1
HLA/Pitch RTI	+1	+1
FAMAS	+1	+1

Table 7.5: Scores for the availability evaluation

7.8 Flexibility

The flexibility quality attribute is concerned with the ability to dynamically joining and leaving of applications during a running session. For the Railway Gaming Suite this means the ability to join new players.

7.8.1 F1 Support new players

Some scenarios require for dynamic joining of new players during a session. For example the train drivers only need to be connected to the system when their trains need to drive. Adding new players has two major consequences. First during play the other participants need to be informed of the new service and start sending messages to and receiving from it. Second the new service must be brought up-to-date with the current state of the system.

Again the loosely coupled nature of SOA supports this requirement together with the discoverable over the network characteristic. In the SOGS architecture services are able to join the running simulation at any time. The ServiceBroker decides when they are able to join, so their is no problem with the consistency of the game. Making sure the new player is up-to-date with the rest of the system is dependent on the implementation of the protocol. State saves can be arranged by a separate service. Score +1.

Pitch RTI is able to join new federates to the system at any time. It does not however have built in functionality for continuous state saves [22]. Adding this functionality poses a risk due to the tightly coupled RTI. It would be possible to make an extra federate that supports this functionality. Due to the uncertainty of successfully making such a federate it gets a score of 0.

The Run Control Subsystem of FAMAS reads the list participating applications from the Scenario Object. It waits for each specified participant to make contact. It assigns port numbers to all federates so they can connect to each other. The simulation start after all participating applications have joined. No new participant can join afterwards. Due to the fact that FAMAS is a research product this could be changed. However since it is not possible in the current implementation it get a score 0.

7.8.2 Flexibility evaluation results

Table 7.6 shows the scores of the evaluation of the extendibility requirements.

	F1
SOA/SOGS	+1
HLA/Pitch RTI	0
FAMAS	0

Table 7.6: Scores for the flexibility evaluation

7.9 Usability

One of the advantages of having a gaming simulation suite is having multiple games and simulators at your disposal to construct different scenarios with. The ability to quickly build and redefine the scenarios results in the following requirements; resetting the system after running a session and setting up a new scenario.

7.9.1 U1 From session-to-session

When using a gaming simulation suite it should be possible to run the same scenario multiple times, or perhaps with some small alterations. The ease to do this enhances the usability of the system. For a session the games and simulators are already connected to each other.

In a service oriented architecture like SOGS the games an simulators can join as they please. By publishing and subscribing to the correct messages a session can run. If all services are set to work together SOGS will take care of the rest and thus there should be no problem running a session again. Score +1.

This requirement is supported by all three architectures. The federation-federate set-up of Pitch RTI as an HLA implementation makes it easy to start the RTI again and connect the federates. The scenario has already been defined so there should be no problem with running a new session. Federates themselves can have different settings, but this should not effect the RTI. Score +1.

FAMAS has an scenario object that contains the information on the games and simulators joining the session. Running the session again does not effect this. Setting up the system should thus not be difficult. Score +1.

7.9.2 U2 Scenarios set-up

Setting up a new scenario requires all games and simulators needed being available. It should then be possible to connect them to the architecture and it should take care of the rest.

All three architecture implementations have been designed to make it possible to connect different games and simulators to it. If all applications needed are available it should not pose a problem to connect them and create a new scenario. For Pitch RTI and FAMAS this is one of the core values of the design. The *interoperability* and *discoverable over the network* characteristics of SOA support this requirement as well. The SOGS architecture which is design with these characteristics in mind shows an implementation example where this works. Score +1.

7.9.3 Usability evaluation results

Table 7.7 shows the scores of the evaluation of the extendibility requirements.

	U1	U2
SOA/SOGS	+1	+1
HLA/Pitch RTI	+1	+1
FAMAS	+1	+1

Table 7.7: Scores for the usability evaluation

7.10 Maintainability

Coupled to the ability and ease to connect new services to the system is the documentation to do this. The extendibility has been handled above, here we look at the provided documentation to do so.

7.10.1 M1 Documentation

Documentation is this case is the information that is available for using the architecture.

In the case of a SOA architecture only the SOGS implementation currently exists and it is poorly documented due to the fact that it is a prototype. No documentation exists for using a SOA solution since there is none, thus the score is -1.

Since Pitch RTI is a commercial product it comes with good documentation. The hands-on experience with the PitchRTI has confirmed this. Score +1.

FAMAS is a research product and the documentation is not at the level of commercial software. Although there is some documentation on the workings of FAMAS, during the experiment implementation this was not enough to easily implement the experiment applications. Score -1.

7.10.2 Maintainability evaluation results

Table 7.8 shows the scores of the evaluation of the extendibility requirements.

	M1
SOA/SOGS	-1
HLA/Pitch RTI	+1
FAMAS	-1

Table 7.8: Scores for the maintainability evaluation

7.11 Summary

In this chapter we have evaluated three architectures based on requirements from chapter 3. The evaluation was done using three methods implementation details, hands-on experience with the architectures and test experiments. Section 7.2 describes the implementation and execution of the experiments to get the test results.

For each of the architectures a score is given per requirement on if the implementation of the requirement would pose are risk to the system. The score are -1, 0 or +1 of risk, borderline or non-risk, respectively.

The results of the evaluation will be used in the next chapter to answer the research questions:

- **RQ2** How well does a service oriented architecture support the requirements of gaming simulation suites?
- **RQ3** How does a service oriented architecture compare to other architectures?

Chapter 8

Discussion

The goal of this thesis is to see if an architecture based on the SOA paradigm is appropriate for gaming simulation suites. The main research question therefore is:

- **RQ** Is SOA a suitable architecture for Gaming Simulation Suites?

In the previous chapter we have seen how well the SOA paradigm supports the requirements we established for gaming simulation suites based on the case study the Railway Gaming Suite. We have also seen how the other two architectures support the same requirements. In this chapter we will discuss the results from the evaluation and compare the architectures to each other. In the table 8.1 the results of all requirements are shown again.

	P1	P2	E1	E2	E3	C1	C2	C3	A1	A2	F1	U1	U2	M1
SOA/SOGS	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	-1
HLA/Pitch RTI	+1	+1	+1	-1	+1	+1	+1	+1	+1	+1	0	+1	+1	+1
FAMAS	-1	+1	0	+1	+1	+1	+1	+1	+1	+1	0	+1	+1	-1

Table 8.1: Results of the evaluation

Next to the requirements we also found five main sensitivity points for gaming simulation suite architectures in chapter 3. These sensitivity points are:

S1 The performance is sensitive to implementation of the architecture.

S2 Making the system synchronized and keeping strict causality comes at a cost of performance.

S3 Increasing reliability comes at a cost of performance.

S4 Adding new players during a running session can disrupt the real-time requirement.

S5 Ease of adding new services, components or ontology is sensitive to the quality of the documentation.

In this chapter we will discuss the results of the evaluation and look at how the sensitivity points are handled by the three architectures. After this we will discuss the threads to the validity of the evaluation we performed.

8.1 Suitability of a Service Oriented Architecture

We start by looking at the second sub research question supporting the main question, which is:

- **RQ2** How well does a service oriented architecture support the requirements of gaming simulation suites?

In the previous chapter we have seen for every requirement how well the SOA paradigm supports it. As we can see in table 8.1 almost every requirement is supported. Supported in this case means that the SOA paradigm does not prohibit the implementation of the requirement or even that the characteristics of SOA directly support the requirement. Thus it is possible to make a service oriented architecture that is suitable for a gaming simulation suite.

Based on the requirement alone we can conclude a service oriented architecture is very well suited for gaming simulation suites. However the requirement evaluation does not expose the influences they have on each other. As stated above we found five main sensitivity points during the ATAM. The sensitivity points expose the risks of requirements in relation to other requirements.

S1 The performance is sensitive to implementation of the architecture.

During this thesis project and the literature study preceding it no service oriented architecture was found specific to gaming simulation suites. This means in order to use a SOA an existing implementation must be converted or a new architecture must be created. The implementation of the architecture has a great impact on the performance. The SOGS architecture shows it is possible to make an implementation that meets the performance requirements we set for gaming simulation suites. Since SOGS is but a prototype that has been implemented in a couple of weeks these results are not conclusive. The prototype does for example not provide extensive reliability functionalities (see below for effects of added reliability). On the other hand it has not been optimized either.

The fact that there is no SOA gaming simulation suite implementation poses a risk and an opportunity. The opportunity is that the requirements of a gaming simulation suite can all be integrated in an implementation from the start. Therefore the result should in theory be a very suitable architecture for gaming simulation suites. The risk on the other hand is the construction of an architecture adds a lot of extra work to a project. In case of the Railway Gaming Suite it is added to the construction of the games and simulators that make up the suite.

S2 Making the system synchronized and keeping strict causality comes at a cost of performance.

In chapter 2 we discussed different methods for synchronization of a distributed system. These methods have in common that they wait for all critical applications to finish before continuing. This means the entire system is as fast as the slowest application. The consequence is that real-time performance requirement might not be met. This effect can not be completely eliminated since we want the system to stay synchronized and strictly causal. There are ways to limit this effect however. For example by excluding some applications from the synchronization if this is not necessary, or just exclude some message types. This could be the case for the location update messages in a dead reckoning mechanism as discussed in chapter 6. If one of these messages is late it will be taken care of by the mechanism itself.

This sensitivity point is a risk for all distributed architectures for gaming simulation suites. The SOGS architecture and HLA RTI have shown it is possible to have a synchronization mechanism in place and still have a high enough performance. The advantage of making a new service oriented gaming simulation architecture is a new mechanism can be implemented that is based on current experience in this field.

S3 Increasing reliability comes at a cost of performance.

Making the system more reliable like in case of the session duration requirement (A1) means certain checks must be performed in order to see if all is well. Examples of this can be: delivery of message checks or checks to see if a service is still connected. This added functionality to the architecture results in extra messages which could effect the performance. The implementation of these checks defines the actual effect. For example check if a service is still up can be done client side by a monitoring application and a message will be send only when the client is down opposed to sending polling checks over the network to each client to see if they are still up. This sensitivity poses a trade-off for all architectures.

S4 Adding new players during a running session can disrupt the real-time requirement.

As stated in the flexibility evaluation in chapter 7 (F1) adding new players has two major consequences. First during play the other participants need to be informed of the new service and start sending messages to and receiving from it. Second the new service must be brought up-to-date with the current state of the system. Informing the current participants of a new service should not disrupt the real-time requirement that much. For example it could be done like in the SOGS architecture, where the new service announces itself at the ServiceBroker, which in turn sends updates to the current participants. The ServiceBroker determines if the request to connect is granted. This requires some additional messages, but tests showed this was no problem.

In order to bring the new service up-to-date there should be some kind of state save system. An extra service could be made to take care of this. However this requires sending messages with a lot of data, which may take some time to be received by the new service. Finally all of this should be done while keeping the system synchronized, so the new service needs to get all messages from the start of the join as well as the state update message. At the service side it should bring the game up-to-date with these messages and allow to player to play the game. Exact protocols to do this have not been researched in this thesis, so no definite answer can be given. Probably a trade-off must be made as to how important the real-time requirement is when a new player joins. For example the system could be paused for a moment to take care of the join and then continue with the added player. Another possibility is to do the state update in small steps meaning the new player has to wait until this is completed. Taking into account the state is continuously updated this might take a lot of time. Of course these steps are extremes and other protocols can be made.

S5 Ease of adding new services, components or ontology is sensitive to the quality of the documentation.

Adding new services, components or ontology has been identified as an important quality attribute of a gaming simulation suite. The developers of a gaming simulation suite are often not the developers of the underlying architecture as well. In order to use the architecture they need to have proper documentation on how to connect services to it and part of how the architecture itself works.

In case of a service oriented architecture for a gaming simulations suite no such documentation exist, since no such architecture exists yet. Here the same risk is exposed as in the first sensitivity point (S1). However this risk has been identified and can be taken into account for building such an architecture and thus be avoided.

The next step is to look at the current architectures and see how well they support to quality attributes.

8.2 Suitability of HLA and Pitch RTI

The evaluation shows that HLA and in particular Pitch RTI has little risk to use as an architecture for simulation gaming suites. This does not come as a surprise since it has been developed for exactly this purpose. HLA scores a -1 and 0 for the new simulation components (E2) and adding new players (F1) requirements, respectively. This is a consequence of the rules of HLA, which make the implementations (like Pitch RTI) less flexible for change. The inner workings are highly interdependent and thus not easy to change or extend. For commercial products, like Pitch RTI, this is stricter then for open source implementations of HLA, like CERTI and PoRTIco (see chapter 6). The open source implementations still need to follow the HLA rules to be a IEEE 1516 HLA compliant implementation. On the other hand, an advantage of using a commercial product is that it is supported by the company that publishes it.

S1 The performance is sensitive to implementation of the architecture.

For HLA there are different implementations. Even though the implementations are restricted to the HLA standard there could still be an effect on the performance. Testing this effect is outside the scope of the thesis. We did however test one implementations of HLA, namely the PitchRTI and from these results we can say the performance requirements are met by this implementation. From this we can conclude that HLA allows for an implementation that can support the performance requirements.

S2 Making the system synchronized and keeping strict causality comes at a cost of performance.

As we have seen from the experiments the PitchRTI implementation is able to support performance requirements while using its synchronization mechanism. This means the synchronization mechanism does not effect the performance in such an amount that it does not support the requirements any more.

S3 Increasing reliability comes at a cost of performance.

The PitchHLA has some reliability check already build into the architecture. This has however not been explored during the experiment tests. As stated above this sensitivity point is a trade-off for all three architectures.

S4 Adding new players during a running session can disrupt the real-time requirement.

The evaluation of the flexibility requirement (F1) (see chapter 7) showed us the PitchRTI is able to join new players at any time, but has not state-save functionality. The actual effect on the real-time requirement is not known, but as we discussed in the SOA section (8.1) it will probably be a trade-off between keeping the game running and adding the new player instantly.

S5 Ease of adding new services, components or ontology is sensitive to the quality of the documentation.

Depending on the implementation of HLA the quality of the documentation is different. Commercial implementation like PitchRTI provide professional documentation that has been very helpful during the implementation of the experiments. For the work package of the RGS project more implementations have been researched. In case of CERTI[1] for example the documentation was not of the level to make for an easy implementation of experiment tests. This is further explored in the work package report[22].

8.3 Suitability of FAMAS Simulation backbone

From the evaluation we see there are some risks involved with using FAMAS as an architecture. To start the tests showed that the performance of FAMAS is not enough to support a gaming simulations suite with services that send a lot of messages. Since FAMAS is a research product this performance issue can probably be improved, but there is no guarantee for this and thus it stays a risk. The fact that FAMAS is a research product also makes it a risk with implementing new connecting applications, since this is not intuitive and not well documented. This comes back clearly in the extendibility and maintenance quality attribute requirements. FAMAS is an open source product makes that it can be used and changed to fit the requirements, but the question is whether this is the best starting point to do so. As it is now it is unable to support the requirements of a simulation gaming suite, in particular the Railway Gaming Suite.

S1 The performance is sensitive to implementation of the architecture.

The experiments have shown us that the current state of FAMAS is unable to support the real-time performance requirement. Since FAMAS is a research product there are

[1] *http://savannah.nongnu.org/projects/certi/*

possibilities to improve on this. The main components are however already built and changing these means changing the architecture. This sensitivity point exposes the risk of using FAMAS as it is.

S2 Making the system synchronized and keeping strict causality comes at a cost of performance.

The experiments did not show us where the performance bottleneck in the implementation of FAMAS is located. It could be that the synchronization mechanism is responsible for this. As stated above the synchronization will always influence the performance since it makes the system as slow as the slowest participant.

S3 Increasing reliability comes at a cost of performance.

For the reliability the same goes as for the other architectures. It is a trade-off between reliability and high performance.

S4 Adding new players during a running session can disrupt the real-time requirement.

Adding new players is not supported by FAMAS in the current version. Components need to exist from the start and there is not state save functionality. This could be added like in the other architectures. This sensitivity point has the same trade-off as the other architectures.

S5 Ease of adding new services, components or ontology is sensitive to the quality of the documentation.

As stated in the evaluation of the maintainability requirement (M1) the documentation of FAMAS is limited. The hands-on experience showed the implementation of a system using FAMAS proved to be very difficult. The quality of the FAMAS Simulation Backbone in its current state is not sufficient to easily add new services, components or ontology. Further development of this architecture can take care of this sensitivity point.

8.4 Threats to Validity

The evaluation we have performed for the thesis report is based on some assumptions we have done. These assumption are a risk to the validity of the evaluation. In this section we explain the risks of taking these assumptions and how we tried to minimize these risks. We have identified three assumptions on which the evaluation is based and we think require extra explanation; the appropriateness of the case study RGS, the evaluation of SOA and the reliability of the experiments.

8.4.1 Appropriateness of the case study RGS.

Throughout this thesis we use the Railway Gaming Suite to base the suitability of a SOA for gaming simulation suite on. The RGS is used as an exploratory case study as described in [25]. It is used to get the quality attributes important to gaming simulation suites. In chapter 3 we provide an explanation for each quality attributes as to how they are important to gaming simulation suites in general. During the ATAM we tried to generalize as much as possible with the information we got from the Railway Gaming Suite. The connection of this thesis with the Railway Gaming Suite project at the Systems Engineering and Policy Organization Law and Gaming departments meant the focus stayed on the RGS. The risk of this approach is we missed important quality attributes for gaming simulation suites.

8.4.2 Evaluation of SOA.

As we can see from the scores from the evaluation, a service oriented architecture supports all quality attribute requirements except the maintainability. Looking at the evaluation it might seem unfair to say something in line of; the SOA paradigm does not prohibit the requirements so it gets a score +1. The reason for these answers is that the goal of this thesis is to see if it is possible to create a SOA that supports the requirements of a gaming simulation suite. The implementation of the SOA is not set. We use the prototype implementation (SOGS) to show that some of the requirements are indeed possible. Building a complete architecture that supports all requirements is outside the scope of this project.

As far as the other two architectures are concerned. For HLA there are specific requirements that make an architecture HLA compliant. This has the effect that many design decisions are set and this prohibits or makes it very hard to implement some requirements. We use the Pitch RTI as an implementation of HLA, since it is compliant to the HLA specific IEEE 1516 Standard. For FAMAS the restriction goes even further because there is only one implementation of the FAMAS Simulation Backbone. The modular way in which FAMAS is built allows for some flexibility, but the main architectural structure is set. Like HLA this restricts the implementation of some of the requirements.

On a side note if we look at the implementation of HLA and FAMAS we see they have a lot of the same characteristic as SOAs, while they are not designed as such. The biggest difference with HLA is that the functionality of the RTI is highly interdependent. The federates themselves are loosely coupled, self-contained and discoverable over the network through the RTI. FAMAS has strived to separate the functionality of the RTI as separate modules.

8.4.3 Reliability of the experiments.

The evaluation is in part based on the results of the experiments we performed. The validity of the experiments is thus important to the overall validity of the evaluation. We have devided the risks to the validity in three part; hardware, implementation and set up of the experiment.

First we start with the hardware. In order to minimize the influence of the hardware used for the experiments we took some precautions. To minimize the risk of the hardware restricting the throughput of the number of messages we used a 1Gb/s switch, new Cat6 network cables, which are suitable for a Gigabit Ethernet and laptops with Gigabit Ethernet cards. Next to this we ensured the laptops were the same for all services we used. The laptops all had the same software image installed on them. The standard network traffic was limited for this image (see appendix C).

Second is the implementation of the experiments. We used the same Sender and Returner applications in all three experiment implementations. Implementing the experiments was however mainly done by three different programmers the three architectures. There was cooperation between the people that implemented the code. We have worked together on the code and discussed the expected results. For the synchronization requirement the importance was made clear and extensively discussed. Different coding methods of the programmers could have influenced the performance of the architectures however.

Finally the set up of the experiment has an influence on the results of the experiments. As we have seen in chapter 7 the variables of the experiments were strictly controlled. This way we could precisely show the conditions under which the results were acquired. By having these restrictions we minimized the chance of other influences on the experiments.

8.5 Summary

In this chapter we discussed the results of the previous chapters. We looked at the evaluation results of chapter 7 for the three architectures. The suitability of the three architectures is separately discussed by looking at the evaluation results of chapter 7 and the sensitivity points from chapter 3. Finally we have looked at the threads to the validity of the evaluation we performed.

Chapter 9

Conclusion and Future work

This thesis report is part of an investigation of architectural paradigms for gaming simulation suites. The focus of this report is to investigate whether service oriented architectures are suitable for gaming simulation suites. The research project at the Systems Engineering and Policy Organization Law and Gaming departments provided a case study to use for this thesis. The thesis project was performed simultaneously with other projects into architectural paradigms which provided more background into gaming simulation suite architectures. Some of this research is used in this thesis. In order to steer our research, we set out to answer the following main research question:

- **RQ** Is SOA a suitable architecture for gaming simulation suites?

In this chapter we will go over the researched questions posed at the start of the thesis and see if they are all answered. We will look at the contributions we made during this thesis, work related to that of this thesis and possible future work based on this thesis.

9.1 Answering the research questions

In this section we take a look at the research questions from chapter 1 again and discuss shortly how they are answered in the chapters.

- **RQ1** What are the architectural requirements to determine the suitability of gaming simulation suites?

To answer this we first need to take a look at the sub questions:

RQ1.1 *What are Gaming Simulation Suites?* In chapter 2 we have given an overview of the definition of gaming simulation suites and have shown some important aspects. In short a gaming simulation suite is a collection simulation games, which is used for training or prediction. We gave and overview of the Railway Gaming Suite which is used as a case study.

RQ1.2 *What is a good method to determine architectural requirements of a system?* Chapter 3 gave an overview of a popular analysis method for software architectures, called Architectural Trade-off Analysis Method. Using this method an analysis was made of the Railway gaming suite in order to find the requirements to determine the suitability of gaming simulation suites.

With this we providing an answer the sub question **RQ1**. In short the ATAM provided us with the following requirements:

- Performance, consisting of real-time play and redundancy.
- Extendibility, consisting of new applications, new simulation components and new ontology.
- Consistency, consisting of time paradigms, causality and synchronization.
- Availability, consisting of session duration and failure detection.
- Flexibility, consisting of support new players.
- Usability, consisting of session-to-session set up and scenario set up.
- Maintainability, consisting of documentation.

These requirements are used to answer the other research questions.

- **RQ2** How well does a service oriented architecture support the requirements of gaming simulation suites?

For this question we needed the result of **RQ1** together with additional information for which we constructed the following questions:

RQ2.1 *What are the principles of a service oriented architecture?* The principles of SOA are explained in chapter 4. It is a distributed network architecture where the nodes are called services. The core values of a service oriented architecture are that services are; loosely coupled, self-contained and discoverable over the network. In the chapter different design approaches for network deployment, message transmission, service discovery and interoperability are described.

RQ2.2 *Can we construct a prototype SOA to test performance requirements?* Chapter 5 gives the design and implementation of a prototype SOA. The prototype is called Service Oriented Gaming and Simulation (SOGS). It is designed keeping the SOA principles and aspects of gaming simulation suites in mind. This prototype is used in tests described in chapter 7. These tests are part of the evaluation. Thus we can say it is possible to construct a prototype SOA to test some requirements.

Using the design principles and approaches from chapter 4 and the SOGS architecture prototype we evaluated whether a SOA is able to support the requirements of gaming simulation suites in chapter 7. The evaluation showed us that a SOA should be able to support the requirements we set, thus a service oriented architecture supports the requirements of gaming simulation suites very well.

- **RQ3** How does a service oriented architecture compare to other architectures?

The third sub question is meant to further investigate the suitability of SOA. In order to answer it we need to answer the following questions:

RQ3.1 *Are there architectures currently in use for gaming simulation suites?* At the start of the work package of the RGS project which this research is part of multiple architectures were identified already. These were HLA, FAMAS and the SOA paradigm. The HLA and FAMAS architectures are existing architectures for gaming simulation suites.

RQ3.2 *What are the architectural approaches of these architectures?* In chapter 6 the design approaches of the HLA and FAMAS architectures are explained. HLA is designed specifically for gaming simulation suites and has been in use for several years. It has a main component called the RTI that provides all the functionality of the architecture. FAMAS is based on HLA but has been set up in a more modular way by separating the functionality of the RTI in different components.

RQ3.3 *How well do the other architectures support the requirements of gaming simulation suites?* The evaluation of chapter 7 has shown us that HLA and in particular the PitchRTI implementation are able to support most of the requirements. The main downside of HLA is that the RTI is tightly coupled and adding new functionality to it is complicated. The FAMAS implementation solves this problem with its modular design, but is unable to meet the performance requirement and due to the fact that it is an existing research project risks are involved with developing it further.

Next to the requirements we also looked at the sensitivity points from chapter 3. These are discussed in chapter 8 and we see the same risks are present for the three architectures. Taking all this into account we can conclude that the SOA paradigm is at least as suitable as the other two architectures and on some points even better.

- **RQ** Is SOA a suitable architecture for gaming simulation suites?

Now we have answered all the sub questions we can return to the main question and provide an answer for it. In short we can say; yes, SOA is a suitable architecture for gaming simulation suites. It supports the requirements we set and compares very well to the other architectures.

There is however no existing SOA implementation. This adds the risk of development of the architecture to development of the games and simulations themselves. Now we have to take into account that no such architecture exists and needs to be designed, developed, tested and maintained. Another factor is that there is an alternative solution, HLA, that supports most of the requirements as well. The work package 1 report [22] came to the conclusion to use the Pitch RTI for this reason.

Using the SOA paradigm to build a new implementation for the ground up provides the opportunity to implement all the requirements for gaming simulation suites in a loosely coupled way and thus decreasing the risks of the tightly coupled Runtime Infrastructure of High Level Architecture. Using experience gained with SOA as an enterprise architecture and HLA as simulation gaming architecture should be a good start to implement a new architecture.

9.2 Contributions

In the context of this thesis we made the following contributions:

- We acquired quality attribute requirements to evaluate the suitability of distributed architectures for gaming simulation suites. These requirements are based on a single case study, namely the Railway Gaming Suite.

- We constructed a prototype SOA called Service Oriented Gaming and Simulation. The prototype includes; dynamic joining of services, fast messaging system based on the publish-subscribe principle, synchronization mechanism, tool to help with building a data model.

- We evaluated three different architectures (SOA, HLA and FAMAS) using these requirements.

- We made a comparison based on the evaluations of the architectures.

- We showed the service oriented architecture paradigm is suitable for gaming simulation suites.

9.3 Related Work

Even though there is no SOA implementation specifically designed for gaming simulation suites, there are implementations that might be used for this. One such implementation is Real Time Infrastructure Data Distribution Service (RTIDDS). This is a commercial product and the company behind it claims it can be described as a real-time SOA. RTIDDS is an implementation of the Object Management Groups Data Distribution Service[1]. DDS is a specification of a middleware for distributed systems that uses a publish/subscribe system. The Getting Started Guide [19] of RTI DDS describes it as follows:

> RTI Data Distribution Service is network middleware for real-time distributed applications. It provides the communications service that programmers need to distribute time critical data between embedded and/or enterprise devices or nodes. RTI Data Distribution Service uses a publish-subscribe communications model to make data-distribution efficient and robust.
>
> RTI Data Distribution Service implements the Data-Centric Publish-Subscribe (DCPS) API of the OMGs specification, Data Distribution Service (DDS) for Real-Time Systems. DDS is the first standard developed for the needs of real-time systems, providing an efficient way to transfer data in a distributed system. With RTI Data Distribution Service, you begin your development with a fault-tolerant and flexible communications infrastructure that will work over a wide variety of computer hardware, operating systems, languages, and networking transport protocols. RTI Data Distribution Service is highly configurable, so programmers can adapt it to meet an applications specific communication requirements.

This description makes it an interesting architecture to research in the context of gaming simulation suites. During this thesis project some time was spent on RTIDDS. A similar test implementation as for the other architectures was made. The problem was that no synchronization mechanism was already in place and implementing it

[1] $http://www.omg.org/technology/documents/dds_{s}pec_{c}atalog.html$

proved to be challenging. With more research however RTIDDS could be a starting point for a Service Oriented Gaming Simulation Architecture.

The comparison of HLA and SOA is not something new and has been researched before [23][18][30]. These research studies however mainly focus on Web Services as an SOA implementation. By combining these terms like Service Oriented High Level Architecture come into play. The latest HLA implementation, HLA evolved, also has a Web Service API to connect Web Services to the RTI. The problem however that the core of HLA, the Run Time Infrastructure, is not designed with the SOA principles in mind. This is not surprising since HLA was already developed before SOA became popular. For this study these HLA-SOA hybrid architectures are not seen as a SOA solution.

9.4 Future Work

The result of this thesis is that the SOA paradigm is suitable for gaming simulation suites. This is a starting point for actually build such an architecture. Implementations of the SOA design approaches can be made and compared to each other in order to find out which is the best suited. Exact performance effects of for example synchronization and reliability checks can be researched. More existing and future gaming simulation suites can be examined in order to further refine the list of quality attribute requirements.

Bibliography

[1] High-level architecture interface specification. version 1.3. Technical report, U.S. Department of Defense, 1998. `http://www.msco.mil/HLAComplianceTesting.html`.

[2] High-level architecture object model template specification. version 1.3. Technical report, U.S. Department of Defense, 1998. `http://www.msco.mil/HLAComplianceTesting.html`.

[3] C.C. Abt. *Serious Games*. University Press of America, 1970.

[4] G. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 2000.

[5] M. Barbacci, R. Ellison, A. Lattanze, J. Stafford, C. Weinstock, and W. Wood. Quality attribute workshops (QAWs), third edition. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2003. `http://www.sei.cmu.edu/library/abstracts/reports/03tr016.cfm`.

[6] S. Benford, J. Bowers, L. E. Fahlén, J. Mariani, and T. Rodden. Supporting cooperative work in virtual environments. *Computer Journal*, 37(8):653–668, 1994.

[7] Ph. Bianco, R. Kotermanski, and P. Merson. Evaluating a service-oriented architecture. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2007. `http://www.sei.cmu.edu/library/abstracts/reports/07tr015.cfm`.

[8] SOA Blueprint. Soa practitioners guide part 1 why services-oriented architecture?, 2006. `http://www.soablueprint.com/yahoo_site_admin/assets/docs/SOAPGPart1.290211145.pdf`.

[9] C. Boer. *Distributed Simulation in Industry*. PhD thesis, Erasmus University Rotterdam, The Netherlands, 2005.

[10] J.O. Calvin and R. Weatherly. An introduction to the high level architecture (HLA) runtime infrastructure (RTI). Technical report, 1996. `http://dss.ll.mit.edu/dss.web/96.14.103.RTI.Introduction.ps`.

[11] T Chatfield. Videogames now outperform hollywood movies. *The Observer*, 09 2009. http://www.guardian.co.uk/technology/gamesblog/2009/sep/27/videogames-hollywood.

[12] P. Clements and L. Northrop. Software architecture: An executive overview. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1996. http://www.sei.cmu.edu/library/abstracts/reports/96tr003.cfm.

[13] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.

[14] DIS Steering Committee. The dis vision, a map to the future of distributed simulation. Technical report, Orlando, Florida, Institute for Simulation and Training, 1994.

[15] J. Färber. Network game traffic modelling. NetGames2002, pages 53–57. ACM, 2002.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[17] HLA Working Group. Ieee standard for modeling and simulation (M&S) high level architecture (HLA) framework and rules. Technical report, IEEE Computer Society, 2000. http://standards.ieee.org/findstds/standard/1516-2000.html.

[18] P. Gustavson, T. Chase, L. Root, and K. Crosson. Moving towards a service-oriented architecture (soa) for distributed component simulation environments. In *Proceedings of the 2005 Spring Simulation interoperability Workshop*, 2005.

[19] Real-Time Innovations. Getting started guide, 2010. http://community.rti.com/docs/pdf/RTI_DDS_GettingStarted.pdf.

[20] L.G. Jones and A.J. Lattanze. Using the architecture tradeoff analysis method to evaluate a wargame simulation system: A case study. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2001. http://www.sei.cmu.edu/library/abstracts/reports/01tn022.cfm.

[21] R. Kazman, M. Klein, and P. Clement. ATAM: Method for architecture evaluation. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2000. http://www.sei.cmu.edu/library/abstracts/reports/00tr004.cfm.

[22] R. Kortmann, S. Meijer, M. Seck, A. Verbraeck, S. Eker, C. Tekinay, and B. Van Nuland. Work package 1: Systems architecture, 2011.

[23] B. Möller and C. Dahlin. A first look at the HLA evolved web service API. In *Proceedings of 2006 Euro Simulation Interoperability Workshop*. Simulation Interoperability Standards Organization., 2006.

[24] M. Papazoglou and W.J. van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[25] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164, 2009.

[26] S. Singhal. *Effective Remote Modeling in Large-Scale Distributed Simulation and Visualization Environments*. PhD thesis, Standford University, Standford, CA, 1996.

[27] J. Smed, T. Kaukoranta, and H. Hakonen. A review on networking and multiplayer computer games. In *multiplayer computer games, proc. int. conf. on application and development of computer games in the 21st century*, pages 1–5, 2002.

[28] R. Smith. Game impact theory: The five forces that are driving the adoption of game technologies within multiple established industries. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2007. `http://www.modelbenders.com/papers/Smith_Game_Impact_Theory.pdf`.

[29] Krissoft Solutions. JMS performance comparison, 2004. `http://hosteddocs.ittoolbox.com/krissoft102904.pdf`.

[30] W.G. Wang, W.G. Yu, Q. Li, W.P. Wang, and X.C. Liu. Service-oriented high level architecture. In *Euro Simulation Interoperability Workshop*. Simulation Interoperability Standards Organization., 2008.

Appendix A

ATAM Phase Two questions and answers

All communication with the user group was done in Dutch. Therefore the questions and answers will also be handled in Dutch here as well. The reason for this is to keep the original answers of the user committee. The questions asked before the meeting where:

- Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
- Wat wordt er gespeeld/gesimuleerd?
- Wie zijn erbij betrokken?
- Welke simulatoren worden er gebruikt?
- Wat zijn de resultaten die u verwacht van de gaming sessie?
- Wat is de opzet van de gaming sessie?

The members of the user committee came up with the following use cases: *Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?*
Toepassing van snelheidssturing, ETMET 2013

Wat wordt er gespeeld/gesimuleerd?
Op basis van gewenst doelcriterium (minimaliseren vertraging en/of max. doorstroming en/of min. energievoorziening) vaststellen of hiervan afgeleide adviessnelheden een betere performance geven bij capaciteitsgebrek door meer treinen of minder infra beschikbaar

Wie zijn erbij betrokken?
Verkeersleiding, planners.

Welke simulatoren worden er gebruikt?
FRISO, TMS

Wat zijn de resultaten die u verwacht van de gaming sessie?
Na de gaming sessies moet duidelijk worden of snelheidssturing een betere performance oplevert.

Wat is de opzet van de gaming sessie?
Informatie over verwachte conflicten en nieuwe planoplossingen wordt aan VL, Trdl (ook mcn?) getoond, prestatie op doelcriterium wordt getoond

Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
Sturen op inhoud van de trein

Wat wordt er gespeeld/gesimuleerd?
De besturing van het treinverkeer wordt gebaseerd op de lading (reizigers of goederen) die in een trein aanwezig is. Van deze 'lading' is bekend waar hij naar toe moet en wat de vertragingsstatus is.

Wie zijn erbij betrokken?
Verkeersleiding, treindienstleiding, planners.

Welke simulatoren worden er gebruikt?
FRISO, SIMONE (of IRIS, een door te ontwikkelen prototype waarin reizigersaantallen en ladinggewicht bekend zijn)

Wat zijn de resultaten die u verwacht van de gaming sessie?
Na de gaming sessies moet duidelijk worden of nieuwe stuurinformatie een betere benutting oplevert en of de taak van VL, DVL, TRDL verandert.

Wat is de opzet van de gaming sessie?
Informatie uit de realisatie/simulatie wordt getoond op de schermen die op de post staan, inclusief nieuwe stuurinformatie waarmee effect van maatregel snel duidelijk wordt.

Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
Is er een alternatief voor de afhandelingsregels (TAD) ?

Wat wordt er gespeeld/gesimuleerd?
De handmatig opgestelde TAD's worden vervangen door automatisch gegenereerde regels

Wie zijn erbij betrokken?
Verkeersleiding, treindienstleiding.

Welke simulatoren worden er gebruikt?

FRISO, SIMONE, Regelgenerator (bijv. SMD-model)

Wat zijn de resultaten die u verwacht van de gaming sessie?
Inzicht in performance van verschillende regelstrategien, noodzaak vooraf opstellen TAD's wordt wel/niet aangetoond.

Wat is de opzet van de gaming sessie?
Prestatie netwerk met verschillende sets voor regels worden beproefd.

Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
Gedistribueerd simuleren van meerdere probleemgebieden in het spoorwegnet

Wat wordt er gespeeld/gesimuleerd?
Elke post (een of meer PPLG's) krijgt informatie uit een netwerksimulatie en stuurt het verkeer binnen het eigen regelgebied. Hoe wordt de ellende uit een gebied overgedragen (of niet) aan een ander gebied?

Wie zijn erbij betrokken?
Verkeersleiding, treindienstleiding.

Welke simulatoren worden er gebruikt?
FRISO, SIMONE

Wat zijn de resultaten die u verwacht van de gaming sessie?
Inzicht in verantwoordelijkheden en regeldoelen op lokaal en bovenlokaal niveau. Welke regelruimte kan je aanbrengen op de verschillende niveaus?

Wat is de opzet van de gaming sessie?
Informatie uit de realisatie/simulatie wordt getoond op de schermen die op de post staan.

Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
Relatie tussen rail/ en transferknelpunten

Wat wordt er gespeeld/gesimuleerd?
Een transferknelpunt kan voorkomen worden door aankomsten van treinen te sturen. De invloed van spreiding in het treinverkeer op transferknelpunten is nu niet bekend

Wie zijn erbij betrokken?
Verkeersleiding, treindienstleiding, planners, adviseurs.

Welke simulatoren worden er gebruikt?
FRISO, SIMONE, SITA

Wat zijn de resultaten die u verwacht van de gaming sessie?
Inzicht in relatie tussen rail/ en transferknelpunten

Wat is de opzet van de gaming sessie?
Uitwerken voorbeeldcases

Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
Ander regime openingen van de Vechtbrug bij Weesp.

Wat wordt er gespeeld/gesimuleerd?
De situatie bij de vechtbrug waarbij gewerkt wordt met vaste brugtijden. In de spelsessie wordt gekeken naar de verschillen in bijsturingsruimte en TDL belasting tussen de huidige situatie en een situatie met flexibele brugtijden.

Wie zijn erbij betrokken?
Het spel wordt gespeeld door treindienstleiders die bekend zijn met het gebied en een spelleider die het spel start/stopt en de rol van brugwachter op zich neemt.

Welke simulatoren worden er gebruikt?
De treindienstleiders spelen de situatie met behulp van de PRL tool. Als simulator wordt FRISO gebruikt.

Wat zijn de resultaten die u verwacht van de gaming sessie?
Na de gaming sessies moet er data zijn over het gedrag van de treindienstleiders tijdens de sessie, in het bijzonder de omgang met de verschillende brug scenario's. Daarnaast moet uit de sessie blijken of het mogelijk is om flexibele brugtijden te gebruiken.

Wat is de opzet van de gaming sessie?
De sessie wordt gespeeld door n treindienstleider achter een enkele computer met vier schermen. De spelleider zit achter een laptop in de buurt van de treindienstleider. De computer is verbonden met de laptop via een LAN.

Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
Verkeersleiding Japanse stijl op een zeer hoogfrequent homogeen net.

Wat wordt er gespeeld/gesimuleerd?
Twee of meer verkeersleiders leiden het zeer drukke spitsverkeer door een corridor. Alleen IC en ST rijdt hier. Zij hebben hiervoor nieuwe middelen om het verkeer te bewaken. De instelling van rijwegen gebeurd volledig automatisch op basis van actueel plan. Bij een verstoring wordt eerst het plan bijgewerkt en dan weer aan de automaten gegeven voor het instellen van rijwegen.

Wie zijn erbij betrokken?
ProRail: VL verkeersleiders (een nieuwe rol)

Welke simulatoren worden er gebruikt?
PRL Game met nieuwe MMI's sterk vereenvoudigde mogelijkheden voor rijwegen instellen (meeste mogelijkheden zijn uitgezet)

Wat zijn de resultaten die u verwacht van de gaming sessie?
Inzicht in werkbelasting van treindienstleiders, machinisten, etc Inzicht in issues bij operationele invoering van tools: bij Verkeersleiding en in de trein

Wat is de opzet van de gaming sessie?
PRL Game met nieuwe MMI schermen (aparte schermen of als pop-up) Nieuwe infra en dienstregeling inladen

Voor welk (toekomstig) ProRail project kan de Railway Gaming Suite gebruikt worden?
Een calamiteiten situatie tijdens ETMET 2015 (Elke Tien Minuten Een Trein- een hoogfrequent dienstregeling)

Wat wordt er gespeeld/gesimuleerd?
De situatie is dat er tijdens spits in 2015 met een ETMET dienstregeling een calamiteit gebeurd. Tijdens de spelsessie wordt er gekeken hoe de verschillende actoren die bij het "product" trein betrokken zijn hun eigen werk uitvoeren en waar de conflicten in de uitvoering zitten.

Wie zijn erbij betrokken?
Het spel wordt gespeeld door de volgende spelers: treindienstleiders, verkeersleiders, machinisten, OCCR, reisinfo, materieelplanners van de vervoerders, algemeen leider

Welke simulatoren worden er gebruikt?
De treindienstleiders spelen de situatie na met behulp van de PRL tool. De machinisten gebruiken de machinsten simulator. Niet alle treinen die in de dienstregeling opgenomen zijn worden met mcn simulator nagespeeld. Als basis dient de FRISO (of een andere simulator) die in verbinding staat met PRL en machinisten simulator. Alle overige spelers gebruiken eigen tools die basis functionaliteiten bevatten die nodig zijn voor hun werk. De vertragingsgegevens en positie van treinen komen uit FRISO (of een andere simulator).

Wat zijn de resultaten die u verwacht van de gaming sessie?
Tijdens de gaming moet er data, beeld en geluid opgenomen worden. Uit de sessie moet blijken of alle processen en werkwijzen goed op elkaar aansluiten.

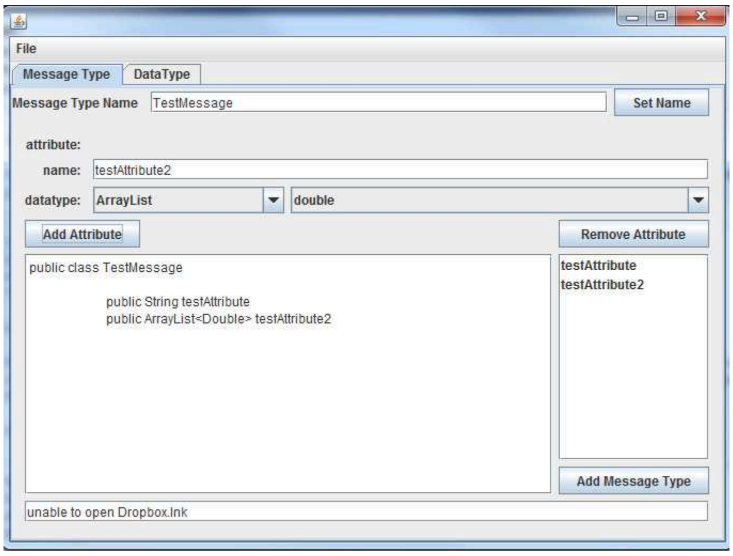
Wat is de opzet van de gaming sessie?
De sessie wordt gespeeld door meer treindienstleiders achter (per trdl) een enkele computer met vier schermen. De machinsten zitten achter een stuurtafel met enkel scherm in een andere ruimte dan trdl's. De spelleider en reisinfo zitten ieder achter een eigen

laptop in de buurt van de treindienstleiders. Reisinfo heeft mogelijkheid om om te roepen. Overige spelers zitten ook in ”eigen” ruimte met een computer/laptop voor zich. Alle computers/laptops zijn met elkaar verbonden via een LAN.

Appendix B

SOGS Data Model Builder tool

The SOGS architecture supports the use of an intermediate language. This language is described as Java Objects. To ease the process of making new Objects for the intermediate language a tool is build. The Objects build with this tool should be send as the data object of the UniqueTimedEvent described in chapter 5. The tool is called the SOGS Data Model Builder (SOGS-DMB). The SOGS-DMB tool is a prototype tool. Here we describe the use and functionality of the tool.



Message Type Name: TestMessage

attribute:
name: testAttribute2
datatype: ArrayList double

```
public class TestMessage

            public String testAttribute
            public ArrayList<Double> testAttribute2
```

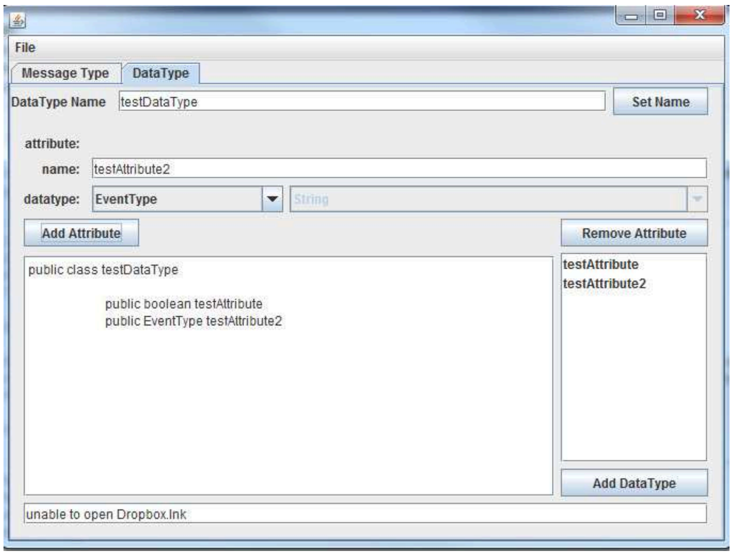
testAttribute
testAttribute2

unable to open Dropbox.lnk

Figure B.1: GUI of a new message type creation with SOGS-DMB

B.1 Using the SOGS Data Model Builder tool

SOGS-DMB is a graphical tool that generates Java classes. The user needs to fill in the name of a new message type and the attributes that are part of the message. For each attribute the object type should be given. This requires some basic knowledge of Java object of the user. Figure B.1 shows the graphical user interface of SOGS-DMB for making a new message type. It is possible the standard data types provided by Java, such as String and Double, are not sufficient to make a new message type. For this the SOGS-DMB tool provides the functionality to make new data types. This works similar to the creation of new message types (see figure B.2). The new data types are



DataType Name: testDataType
name: testAttribute2
datatype: EventType | String

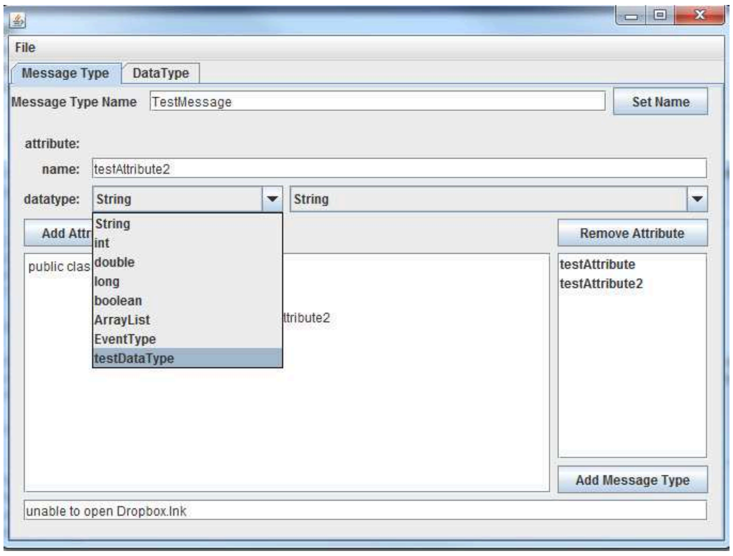
```
public class testDataType

          public boolean testAttribute
          public EventType testAttribute2
```

testAttribute
testAttribute2

unable to open Dropbox.lnk

Figure B.2: GUI of a new data type creation with SOGS-DMB

directly added to the data type lists of the message and data tabs as is shown in figure B.3. The ‘Add MessageType’ and ‘Add DataType’ buttons complete the construction



Message Type Name: TestMessage
name: testAttribute2
datatype: String | String

public clas ... ttribute2

testAttribute
testAttribute2

unable to open Dropbox.lnk

Figure B.3: GUI of a new data type creation with SOGS-DMB

of the new Java classes. It generates a new file with the name of the new message type and the ‘.java’ file extension. For the TestMessage from figure B.1 the corresponding file looks as follows:

```
import java.util.ArrayList;
import java.io.Serializable;

public class TestMessage implements Serializable {

  private static final long serialVersionUID = 1L;
  private String name = "TestMessage";
  public String testAttribute;
  public ArrayList<Double> testAttribute2;

  public TestMessage(String testAttribute,
      ArrayList<Double> testAttribute2) {
    this.testAttribute = testAttribute;
    this.testAttribute2 = testAttribute2;
  }

  public String getName() {
    return this.name;
  }

  public String toString() {
    String result = "TestMessage: ";

    result += this.testAttribute + " ";
    for (Double item : this.testAttribute2) {
      result += item.toString() + " ";
    }

   return result;
  }
}
```

The SOGS-DMB only functions as a generator for Java code. The programmer is responsible for the translation of the application specific data into the message objects created with SOGS-DMB. The tool allows for the creation of new files, opening and adaptation of files and refactoring. The refactoring of a file can be done when the tool itself has been changed, for example when the generation of the toString() method has changed. By using the refactor option the file is opened and displayed in the GUI, but it is also automatically rewritten to a '.java' file using the updated conversion of the tool itself.

Appendix C

Experiment Laptop Settings

We tried to limit the network traffic to the bare minimum. This required to change the standard setting of the Ethernet card and limiting the services running on the laptop.

C.1 Ethernet Card

Settings of the on-board Ethernet card:

```
Broadcom NetXtreme 57xx Gigabit Controller Properties
Driver Date 5-6-2007
Driver Version 10.39.0.0

802.1p QOS Disable
Flow Control Auto
Speed % Duplex Auto
Wake Up Capabilities Both
```

C.2 TCP view

Overview of the TCP endpoints on the laptop:

```
alg.exe      552  TCP cps-dell  1026          cps-dell 0 LISTENING
lsass.exe    772  UDP cps-dell  isakmp * *
lsass.exe    772  UDP cps-dell  4500 * *
svchost.exe  1012 TCP cps-dell  epmap          cps-dell 0 LISTENING
svchost.exe  1132 UDP cps-dell  1025 * *
svchost.exe  1132 UDP cps-dell  ntp * *
svchost.exe  1288 UDP cps-dell  1900 * * 3 399
svchost.exe  1288 UDP 169.254.24.80 1900 * *
svchost.exe  1132 UDP 169.254.24.80 ntp * *
System       4    TCP cps-dell  microsoft-ds   cps-dell 0 LISTENING
System       4    UDP cps-dell  microsoft-ds * *
```

C.3 Network Traffic

A capture has been made of the traffic on the network after disabling of unnecessary services. Capture was made using WireShark [1]. The capture lasted ten minutes. The last four seconds are shown below to illustrate the traffic. These messages were repeated over and over.

```
No.     Time        Source               Destination                    Protocol
453     599.850802  HewlettP_32:28:fe    Spanning-tree-(for-bridges)_00 STP
Info
RST. Root = 32768/0/00:15:60:32:28:00  Cost = 0  Port = 0x8002

Frame 453: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
IEEE 802.3 Ethernet
Logical-Link Control
Spanning Tree Protocol

No.     Time        Source               Destination          Protocol
454     600.926022  HewlettP_32:28:fe    LLDP_Multicast       LLDP
Info
Chassis Id = 00:15:60:32:28:00 Port Id = 2 TTL = 120 System Name = Uitleen Switch

Frame 454: 176 bytes on wire (1408 bits), 176 bytes captured (1408 bits)
Ethernet II, Src: HewlettP_32:28:fe (00:15:60:32:28:fe), Dst: LLDP_Multicast (01:80:c2:00:00:0e)
Link Layer Discovery Protocol

No.     Time        Source               Destination                    Protocol
455     601.850623  HewlettP_32:28:fe    Spanning-tree-(for-bridges)_00 STP
Info
RST. Root = 32768/0/00:15:60:32:28:00  Cost = 0  Port = 0x8002

Frame 455: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
IEEE 802.3 Ethernet
Logical-Link Control
Spanning Tree Protocol

No.     Time        Source               Destination          Protocol
456     602.851084  HewlettP_32:28:fe    HP_09:13:a6          EEE802a
Info
OUI 0x080009 (Hewlett-Packard), PID 0x0003

Frame 456: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: HewlettP_32:28:fe (00:15:60:32:28:fe), Dst: HP_09:13:a6 (09:00:09:09:13:a6)
IEEE802a OUI Extended Ethertype
Data (41 bytes)

0000  02 00 02 00 00 10 d0 00 15 60 32 28 00 7a 21 4f   .........‘2(.z!O
0010  03 93 98 b5 0d 17 88 76 04 71 f2 ae d5 a7 45 62   .......v.q....Eb
0020  ae 00 00 00 00 00 00 00 00                        .........
```

[1] $http://www.wireshark.org/$