A method and framework to evaluate system architecture candidates on reliability criteria

3TU. School for Technological Design

STAN ACKERMANS INSTITUTE

A method and framework to evaluate system architecture candidates on reliability criteria

Konstantinos Filippidis

2015

TU Delft
Delft University of Technology

TU/e Technische Universiteit Eindhoven University of Technology

UNIVERSITY OF TWENTE.

A method and framework to evaluate system architecture candidates on reliability criteria.

Konstantinos Filippidis
September **2015**

A method and framework to evaluate system architecture candidates on reliability criteria

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

Philips Lighting

Eindhoven University of Technology

Steering Group

Henk Stevens
Tanir Ozcelebi
Konstantinos Filippidis

Date

September 2015

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 7.090, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31402474334

Published by
Eindhoven University of Technology
Stan Ackermans Institute

Printed by
Eindhoven University of Technology
UniversiteitsDrukkerij

ISBN
A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-444-1375-5
(Eindverslagen Stan Ackermans Instituut ; 2015/037)

Abstract
Philips Lighting is moving from a lighting component business towards lighting solutions business in professional environments, offering lighting solutions for energy saving, productivity and effect creation. In these solutions (consisting of light sources, sensors and control devices) networked based systems are essential and software makes it possible to add intelligence into the system. Reliability aspects at system level are getting more important and architectural analysis is done during brainstorm sessions to evaluate possible system architecture candidates on reliability criteria. It is crucial that the reliability of the architectures is evaluated more formally, for example based on a model of each of the architectural candidates. This report describes the design, the implementation, and the experimentation with such a method that predicts the reliability and other non-functional criteria of the architecture candidates. Lighting entities, their attributes, behavior, and relationships are the core of evaluating a system on reliability criteria. This approach relies heavily on modeling principle. The results of the project show whether such an approach can been used to determine and evaluate an architecture candidate, and give input on how architecture evaluation can be done in the future.

Keywords
Software architecture, Reliability, Evaluation, Component based architecture, performance evaluation, system architecture evaluation, Performance, Software component, CBSE, Prediction, Modelling, Measurement

Preferred reference
A method and framework to evaluate system architecture candidates on reliability criteria, SAI Technical Report, September 2015. (978-90-444-1375-5)

Partnership

This project was supported by Eindhoven University of Technology and Philips Lighting.

Foreword

How do we take care that we have selected the right System Architecture to fulfill the needs of the application? Different architecture may be thought of or variations within a certain architecture. An architecture needs to be defined in an early stage of a project. So validation of the architectural choices is an important aspect, as modifications on architectural level may be costly at a later stage in the project. The starting point for an Architect are the user requirements and related quality criteria are clear. The architect need to validate them in the possible architectural compositions, and to compare architectural compositions, to select the best one.

Architects may use own experiences to evaluate an architecture, or use the whiteboard to do some rough calculations. But it will take time to evaluate each architectural composition on the related criteria. A tool for an architect, which he can use to quickly model the architectural compositions, to model the usage of the system, and model the actual resources, would be very welcome. The tool should provide enough information on the quality and performance aspects of the architecture for a good first analysis.
The architectural modelling is relevant as it gives guidelines to the architect what is needed to do a first analysis. The architect needs to think on usage of the system and resource needs, to feed the model, and to get some useful results.

Konstantinos made clear to the stakeholder that such a tool is relevant and will help the architects to better understand the architectural choices. After some research on tools and methods Konstantinos proposed to use the Palladio tool. He put a lot of effort in understanding the tool and how to make the typical architectural compositions for Lighting Applications. He experienced that the tool was causing some problems, either within the definition of the models or the composite structure its elves. Maybe we selected the wrong tool? But Konstantinos was convinced that it was the right choice, and came up with some interesting “design patterns” for the architectural composition. When looking back, the learnings of the project is that it is not only about the tool, but by modelling the architecture like this, the architects are enforced to get information on the table on system usage and resource needs, to do a better evaluation. Konstantinos expressed that to us and to the stakeholders, many times.

Henk Stevens

14 September 2015

Preface

This report presents the “a method and framework to evaluate system architecture candidates on reliability criteria” project that was carried out by Konstantinos Filippidis at Philips Lighting HTC, Eindhoven, The Netherlands. The project was conducted as a full-time, nine-month graduation assignment in the context of a two-year technological designer program in Soft-ware Technology popularly known as the “OOTI” program. This post-master program is offered by the Eindhoven University of Technology under the auspices of the Stan Ackermans Institute.

The project’s goal is to evaluate the author as a software designer, while providing Philips Lighting with a proposal framework that will be used by architects when making decisions on which architecture candidate to apply on a new system. This report contains the design solution as well as a description of the process that led there. Therefore, in addition to the new design, the domain, project management conclusions, and retrospective are explained in corresponding chapters.

Konstantinos Filippidis
14 September 2015

Acknowledgements

This project could not have been completed if it were not for the help of a large group of people. Therefore, I would like to thank all people that have contributed to my project and I would like to take the opportunity to thank the following people explicitly.

Within Philips Lighting I was in very close contact with my supervisor. I want to thank Henk Stevens for providing continuous support, feedback, and guidance throughout the project. He assisted me in getting familiar with the lighting domain and also in developing personally as a software engineer. I would also like to thank Rob van Twist, for his contribution in the project and the feedback, especially during the mid-period of the project, he gave a lot of information and support to continue this project. I would also like to thank Berkvens Winfried, for the help and support in the first phase of the project. His input was valuable.

From the university side I am grateful to my supervisor, Tanir Ozcelebi, for the support and steering he provided in the duration of this project. I also would like to thank the director of the PDEng Software Technology program, Ad Aerts and the scientific director Johan Lukkien for giving me the opportunity to be part of the program, and the program's secretary, Maggy de Wert, for assisting with all the issues regarding me and my colleagues for the past two years.

I would also like to thank all my Software Technology colleagues for all the feedback, support and nice memories during our time together.

Additionally, I would like the express my gratitude to my parents, my brother and my friends in Greece for their support regardless of the distance. Moreover, I would like to thank my close friends here in Eindhoven, Chara, Lena, and Antoni, for their continuous support these nine months of the project.

Finally, I want to thank all others that I failed to mention for any kind of contribution throughout the nine months of the project.

14 September 2015
Konstantinos Filippidis

Executive Summary

Currently, when implementing a new system at Philips Lighting, software architects need to evaluate the architecture candidates for that system based on their past experience and by using PUGH analysis. This evaluation process is inadequate in two respects:

- It fails to give an accurate form of the architecture candidate for a system. Past experience can give some insight on how some parts that were used previously can behave; however, it is based only on some knowledge that only architects have

- It is based on assumptions. PUGH analysis matrix is constructed in such way that every element of the architecture candidate has a weight. In the end, the best balance for between the different elements of the matrix is decided, and that is ow the architecture design for the system is defined.

This report describes a project to design that process in a more automated way. All requirements for the new system are acknowledged when introducing an architecture candidate, however, gives the opportunity to model in such way to provide simulation results of that candidate.

The basic concept is to define a use case scenario from the architecture candidate. This provides the parts (components) need to be modelled. A step further is to take the components used in that scenario and modeled them, in a similar way to UML modelling. Furthermore, giving specification for each component is part of the process. This part introduces the functions and the resources that every component uses. Additionally, a composite structure is implemented (which is the architecture candidate for the evaluation), showing the connection and communication between the different components.

But this is not enough, resource environment modeling is needed. In order to provide simulation results, and more specifically performance results, specification of resource environment, such as CPU and HDD, needs to be defined for the system.

Finally, the usage model for the system, is defined. The latter includes triggering events for the system, and what enables the system to run.

By implementing all the above inside the tool, the architect is now able to make simulation tests and gets results on the performance of a proposed system, more specifically, on the architecture candidate proposed. Moreover, the architect can make changes on every step of the modeling and run new simulations.

The process that is described in this report provides the architect with an environment to model all the parts that are crucial to evaluate an architecture candidate. From a matrix in excel sheet we moved to a tool based modeling environment that allows a more automotive process for deciding an architecture to follow.

In the end, the architect is the one that will make the choice of the most suitable architecture to implement, however, now he experiments and simulates the different candidates and he can actually see simulation results on the performance of each one.

Table of Contents

List of Figures

[*All figures in the report should be numbered and should have an appropriate title.*]

List of Tables

[*All tables in the report should be numbered and should have an appropriate title.*]

1.Introduction

Abstract –This chapter presents the main motivation idea of this project, which is the implementation of a framework to evaluate architecture candidates based on reliability criteria. Introductory concepts are presented, as well as the project scope.

1.1 Context

Philips Lighting is moving from a lighting component business towards lighting solutions business in professional environments (e.g., office buildings, shops), offering lighting solutions for energy saving, productivity, and effect creation. In these solutions (consisting of light sources, sensors, and control devices) networked based systems are essential and software makes it possible to add intelligence into the system.

The networked based intelligent lighting systems are complex systems, in which software has a dominant role. Reliability aspects at system level are becoming more important and architectural analysis is done during brainstorm sessions to evaluate possible system architecture candidates on reliability criteria. It is crucial that the reliability of the architectures is evaluated more formally, for example, based on a model of each of the architectural candidates. A method is required that predicts the reliability and other non-functional criteria of the architecture candidates.

1.2 Introductory Concepts

1.2.1. Software architecture

Software Architecture (SA) has been attracting more and more attention from researchers, since the early periods of the last decade of the 20th century. Complexity and demand for quality in software systems are two of the most crucial issues that have led to the increase in the interest in this sub-discipline of software engineering.

Although there are numerous definitions of what software architecture is, we can conclude that the core of all of them is the notion that the architecture of a system describes its gross structure [1]. This structure illuminates the top-level design decisions, including how the system is composed of interacting parts, where the main pathways of interaction are, and what the key properties of the parts are. Additionally, an architectural description includes sufficient information to allow high-level analysis and critical appraisal.

A good architecture can help ensure that a system satisfies key requirements in areas such as performance, reliability, portability, scalability, and interoperability. As a consequence to that, a bad architecture can be disastrous.

1.2.2. Software Reliability

Software reliability is defined by ISO 9126, as “The capability of the software product to maintain a specified level of performance when used under specified conditions”. In other words, software reliability is the probability of a failure-free operation of a software system for a specified period of time in a specified environment. This standard states three key components of reliability: recoverability, fault tolerance, and maturity. Recoverability is “The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure”. Fault tolerance is “The capability of the software product to main-

tain a specified level of performance in cases of software faults or of infringement of its specified interface". Maturity is "the capability of the software product to avoid failure as a result of faults in the software".

In practice, architects may have different architectural compositions, which need to be evaluated, with limited time, to go through a full reliability scenario. Systems consists of hardware and software parts. Reliability can have impact on both; however, system architecture evaluation aims at already improving reliability of software architectures during early development stages. Especially when we are talking about component based systems, the latter will help the software architects to determine the software components mostly affecting the system's reliability, the sensitivity of each component is reliability towards the system's reliability, and in the end, support decisions between different architecture candidates.

1.2.3. Component-Based Software development

The term component-based in software development is used to describe the process of assembling components, in order for them to interact as intended. This approach is mainly used to improve final product quality and to improve software maintainability and reliability.

In addition to component-based software development, when it comes to reliability there are two aspects to be considered:

- Reliability of a component: Reliability assessment of the components and how reliability models are affected by different component usage.
- Reliability of a component based system: Reliability assessment of an application developed using software components as their building blocks.

1.3 Project Scope

The project focused on implementing a framework to evaluate architecture candidates that supports architects on the decision when it comes to choosing a project's architecture.

When it comes to architects perspective of architecture, two things are important:

- Software architecture that reflects the structure of the software product.
- Software architecture that is as simple as possible.

The scope of this 9-month project is to implement a framework that should include:

- method to evaluate the proposed architectures using identified components
- adaptation of properties in the model based on non-functional requirements
- execution environment to run (repeatable) test scenarios for reliability as an objective judgment of architectures

The project is not about creating a complete tool of evaluating architecture models. However, the framework should provide results that give information on whether to create a complete tool at a later stage. Architecture is both related to software and hardware, when it comes to reliability issues. Evaluation of an architecture is done manly on system level, and there both software and hardware parts play key role. For this project, the hardware part was out of the scope.■

2.Stakeholder Analysis

Abstract –The previous chapter gave some insight about the problem statement, while this chapter attempts to analyze the people that are mostly involve in this project and affected by it.

Traditional software development has been driven by the need of the delivered software to meet the requirements of users. Although the definition of the term user varies, all software development methods are based around this principle in one way or another.

However, the people affected by a software system are not limited to those who use it. Software systems are not just used: They have to be built and tested, they have to be operated, they may have to be repaired, they are usually enhanced, and of course they have to be paid for. Each of these activities involves a number – possibly a significant number – of people in addition to the users.

Each one of these groups of people has their own requirements, interests, and needs to be met by the software system. We refer collectively to these people as stakeholders. Understanding the role of the stakeholder is fundamental to understanding the role of the architect in the development of a software product or system.

2.1 Introduction

A stakeholder of a project is an individual, team, organization, or classes thereof, having an interest in the realization of the system. Most system development projects include representatives from most if not all of these stakeholder groups, although their relative importance will obviously vary from project to project.

Stakeholders can affect or be affected by the organization's actions, objectives, and policies. Some examples of key stakeholders are creditors, directors, employees, government (and its agencies), owners (shareholders), suppliers, unions, and the community from which the business draws its resources.

Not all stakeholders are equal. A company's customers are entitled to fair trading practices but they are not entitled to the same consideration as the company's employees.

For my project I can identify two major categories of stakeholders: Philips stakeholders and Tue stakeholders. Both categories have their interest in the project but with different perspectives. Both want to have a successful project in the end; however, their focus towards the procedure to follow in order to achieve the end result is the main difference.

Table 1 – Stakeholders

Stakeholder	Role	Responsibility
Henk Stevens	Project Owner	• Providing architecture vision and high level information. • Overlooking the progress of the project and offering guidance.
Tanir Ozcelebi	Tu/e supervisor	• Overlooking progress of the project, offering guidance, deal with stakeholders' satis-

		faction, planification, and writing the final report.
Berkvens Winfried	System Architect	• Provide pilot project to follow in order to implement mine. • Providing guidance when it comes to architecture decisions.
Peter Fitski	System Architect	• Overlooking the progress of the project, and at later point join the experiments.
Mark Verberkt	System Architect	• Overlooking the progress of the project, and at later point join the experiments, when results are promising.
Rob van Twist	System Software Architect	• Overlooking the progress of the project, and at later point join the experiments. • Offering tutoring on the LM-IP architecture. • Define example scenario to use for the experiments. • Provide data information for the experiments.
Dick Schenkelaars	Quality Systems Manager	• Overlooking the progress of the project, from a quality perspective.

2.2 Stakeholders Categories

As mentioned previously there are two stakeholder categories for the System Architecture Evaluation project. This section defines these two categories.

2.2.1. Philips Stakeholders

The first category refers to the Philips Lighting employees that connect to the System Architecture Evaluation project. We can divide these stakeholders into subcategories, because stakeholders are not equal by belonging to the same category, it does not mean that they have the same perspectives towards the project.

The project owner focuses more on the end result, meaning that the project is successful as long as it meets the requirements for an implemented system that is running. However this is not the case when it comes to the system's architect's requirements, which are how they can benefit from the provided tool, what choices will be made at the implementation phase of the tool, and why this tool will help them when it comes to decision making.

Quality managers are interested in having an assessment method, which justifies the choices made by the architect. They are more interested in the process and the correctness of the assessment.

2.2.2. TU/e Stakeholders

The other stakeholder category concerns TU/e. The supervisor from TU/e, is mostly concerned about the whole procedure towards the project solution. Although he interested in having a functional system in the end, he is mostly concerned with every

step that is made. To be clearer about the latter, risk management, stakeholders' management, project outline, action points and everything that is done is in his interest.■

3.Problem Analysis

In this chapter we define the problem that we intend to solve with the tool that we are going to use. Previously we mentioned the stakeholders, and what they require from our solution. However, we have not mentioned a description of the actual problem.

Problem analysis is the process of understanding real-world problems and user needs and proposing solution to meet those needs. The goal of problem analysis is to gain a better understanding of the problem being solved before development begins.

3.1 Context

Abstract –Philips Lighting is moving from a lighting component business towards lighting solutions business in professional environments (e.g., office buildings, shops), offering lighting solutions for energy saving, productivity and effect creation. In these solutions (consisting of light sources, sensors and control devices) networked based systems are essential and software makes it possible to add intelligence into the system.

The networked based intelligent lighting systems are complex system, in which software has a dominant role. Reliability aspects at system level are getting more important and architectural analysis is done during brainstorm sessions to evaluate possible system architecture candidates on reliability criteria. It is crucial that the reliability of the architectures is evaluated more formally, for example based on a model of each of the architectural candidates. A method is required that predicts the reliability and other non-functional criteria of the architecture candidates.

The quality of an architectural design is critical for successful development of a system. Two most commonly reasons are:

1. Architecture design sets the foundation for the successful achievement of quality requirements

2. Architecture design helps to deal with the ever-increasing complexity of today's embedded systems.

Decisions made in the architecture design phase are aspects that have a very large impact on the cost and quality of the final system. An additional aspect, is that quality requirements can often conflict with one another and with economic constraints.

The growing complexity of today's embedded systems is a factor that mainly induced by customers requiring more and more functionality. Another factor is that the design of embedded systems is presently moving from standalone and partitioned systems to functionally integrated architectures, which are characterized by extensive sharing of information and hardware resources. In such architectures, shared processors and communication channels allow a large number of configuration options at design time and a large number of reconfiguration options at runtime.

Component-Based System development is a promising approach when it comes to developing large and complex systems, while providing efficiency, reliability, and maintainability. This approach allows the software architect to rely on the composed structure, which is not only crucial for the functional requirements but also non-functional properties and software quality as well.

3.2 Roadmaps

Philips as a company is divided in three major domains, Lighting, Healthcare, and Consumer Lifestyle, thus has different perspectives when it comes to roadmaps. This section describes these perspectives, mostly focusing on Philips Lighting. First we describe the Business Roadmap followed by Technology and Product Roadmaps.

3.2.1. Business Roadmap

Since Philips focuses on the three domains, mentioned previously, it is crucial for all domains to strive to capture highly attractive market opportunities. Their Business Roadmaps are going towards that in order to achieve their goals.

Philips Healthcare focuses on the HealtTech opportunity, by serving the Health Continuum and leveraging strengths of Healthcare and Consumer Lifestyle. Healthy Living, prevention, diagnosis, treatment, recovery, and home care are the main business roadmaps of Philips Healthcare domain.

Philips Lighting focuses on lighting opportunity, by offering stand-alone Lighting solutions. Their target business groups are:

- Light Sources and Electronics
- Consumer actuators
- Professional Lighting Solutions

What can easily be observed at Philips Lighting is the movement of the business from a traditional lighting producer towards digital lighting and related services. On the technology roadmap, it is described the focus on new technologies to provide lighting solutions. Their strategy of connected lighting captures the attractive value of lighting solutions. Their aim is to improve people's lives by delivering unique value and energy efficient solutions to consumers.

By following these roadmaps Philips wants to achieve:

- higher growth and profitability
- improved customer focus in attractive markets
- faster decision making
- options for capital market access for Philips Lighting.

Figure 1 – Philips Lighting well positioned to capture growth opportunities

3.2.2. Technology Roadmap

For this section, our main focus is Philips Lighting. Their technology roadmap is formulated of the world's need. There is need for more light, more energy efficient light, and more digital light. These needs have led their technology map on the LED technology to progress by 34% per year. This can be seen on the LED components, LED lamps and modules, LED actuators, and as a result, this reflects on the lighting systems and services.

Moreover, their LED research and development department investments led to game changing innovation and technology revolution.

- Capture high value in the market through High Power products.
- Leverage High Power capabilities into the Mid Power LED.
- Shape the transformation to digital lighting.

Although new technologies are always good to be implemented, another thing that is in the focus of their technology roadmap is performance issues, mainly on operational performance, which has improved supported by strong growth margin management and structural cost savings. Digital lighting is IP based and connected Lighting, which enables the path for Systems and Services. Collecting data via sensors, being part of the digital infrastructure of lighting, will give the opportunity to analyze the data and provide services to improve lighting conditions, energy consumption, or building performance. Connected lighting will allow other domains, such as HVAC, to interact with the lighting system in a seamless way.

3.2.3. Product Roadmap

The previous section is about technology roadmaps. As long as technology is improving, the results can be reflected in the products Philips Lighting is implementing.

Connected intelligent lighting is an area of future innovation in office as well as in home application.

Their product roadmap now is to provide services, not just lighting or application, but both. Taking for example the home automation, this can contain lighting, security, maintenance, and actually that is what they want to provide to the customer, everything as a service and not just a stand-alone application. In the technology roadmap section, it was mentioned that Digital lighting enables the path for Systems and Services. Home automation is a very strong example of what Philips wants to provide to

customers. Finally, we may conclude that the company wants to achieve more services than products in the future, which will add more value and more money.

4.Domain Analysis

Abstract –This section describes background information that has been gathered about events in organizations and how they are handled. This information is to be used to guide the development of software to automate the process of evaluating architecture candidates.

4.1 Introduction

For every system that is implemented inside Philips Lighting, there might be several architecture candidates. Figure 2 shows the procedure as it is currently at Philips Lighting when a new architecture is proposed.

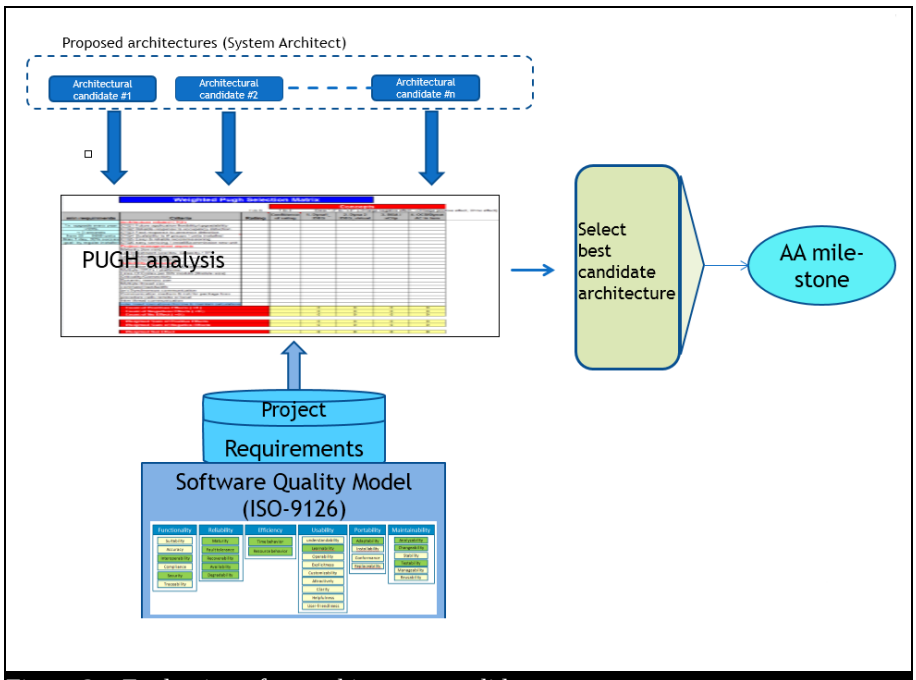

Figure 2 – Evaluation of an architecture candidate

The goal of the proposed evaluation framework is to help the user come up with the best architecture solution, by allowing the user to model parts of each candidate and run simulations to gather measurements.

Using this application, a user will be able to obtain different types of results based on the measurements he chooses to have. Figure 3, shows how the evaluation of an architecture candidate is done with the proposed methods, and where actually it is going to be placed in the current process. The outcome of this process is the Architectural Accepted milestone. For the evaluation process, all related architectural questions and issues should be clarified and reviewed with the product development team. At the end of the process, the result should reflect all the decision that have been made on the accepted 'quality'.

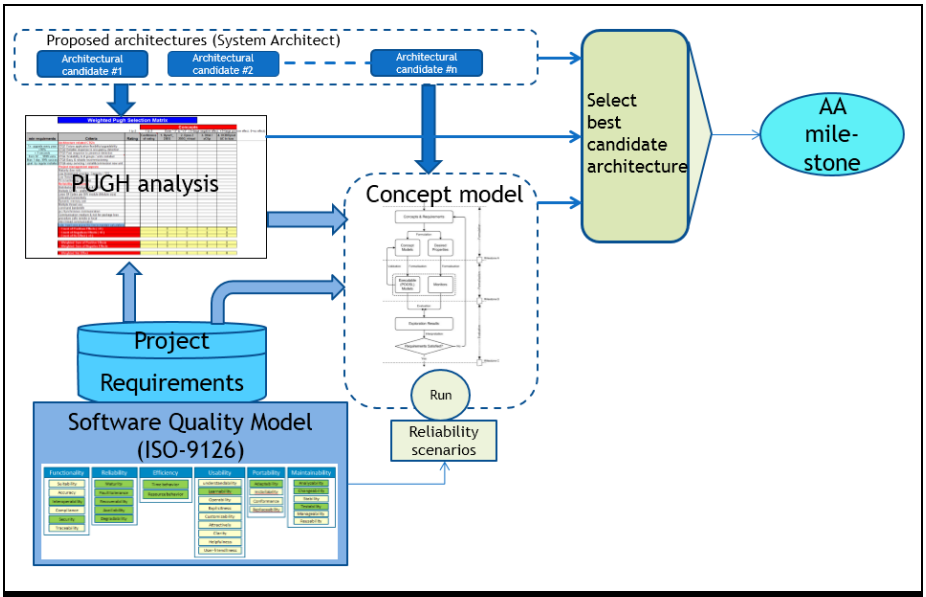

Figure 3 – Evaluation of an architecture candidate with the proposed framework.

Architects are the main users; however another type of user could be someone who can model part of the architecture candidate, before the simulation starts.

4.2 General knowledge about the domain

- Architecture is one of the most crucial parts when it comes to implementing a new system.
- Architects have brainstorming sessions to figure the best option for architecture.
- In these sessions the architecture candidates are analyzed.
- System requirements play a significant role in choosing the best approach.
- Quality requirements are crucial for architecture decisions.

4.3 Clients and users

Potential users of the system would be the architects of Philips Lighting.

- The system's primary goal is to help the architects make a better selection when different architecture candidates exist.
- Architects will be the ones who benefit the most; however they are not the only users.
- Quality person, can be anyone who wants to use it in order to model a specific architecture and make some measurements from it. This user can also be someone who just helps an architect to make the model inside the tool.

Figure 4, shows the use case of an architect that uses the proposed framework.

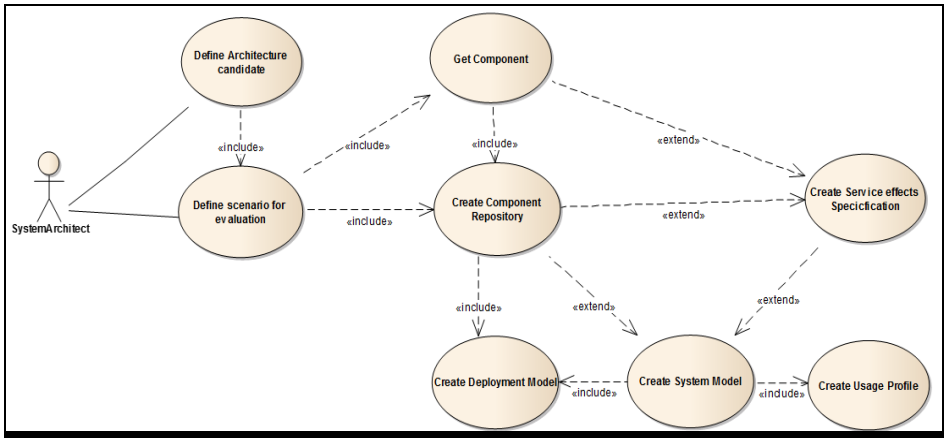

Figure 4 – System Architect Use case

4.3.1. User Stories

This section provides use cases of the proposed system, to help understanding the actual purpose of such a tool. Before the cases, there is the description of what an architect should provide to the system, and, later on, what to expect as output.

A software architect should be able to provide to the system components, intended behavior of the system, structure, resources that the systems needs and the usage scenario.

Once the architect provide such information, then he/she could work on the following inside the tool:

1) Differentiate behavior
   A software architect can change the behavior of a component in order to see different measurements, e.g. implement a method once in actuator and once in a controller.
2) Differentiate composite structure.
   A software architect, can experiment with different composite structures with the same components to examine the performance.
3) Differentiate usage scenario
   A software architect, can define different number of events that trigger the usage scenario of the system.
4) Measurements
   a. Utilization of resources
   b. Histograms on prediction vs simulation

4.4 The environment

All actors have a computer on their desks; it is most common for this to be MS-Windows based, but a significant minority of potential clients use other platforms. Since it is an Eclipse[1] based system that will not cause potential problems.

The tool is a different instance of Eclipse Software development kit, and a potential user only needs to install it on their computer in order to be able to use it.

1 Palladio-Bench release version 3.5.0.

4.5 Tasks and procedures currently performed

The following section describes what can be done inside the system.

- Create component repository, where the different parts of the system are first defined.
- Create service effect specification (SEFF), where behavior is added to the different components of the system.
- Create system model, where the composite structure is defined, to see the whole system interactions.
- Create deployment model, where we define the number resources for our system, such as hard drive and CPU.
- Create usage model, where we model the interaction of the user with the system.

4.6 Competing software

Several approaches for evaluating architecture exist; however not all of them are implemented, and there for not many software tools are available to evaluate an architecture. Similarities exist in all methodologies especially when evaluation is done based on quality attributes. However, the most difficult part in all the systems that exist is how to model quality criteria in order to run simulation and gather measurements, when it comes to evaluating a potential architecture candidate. ■

5.Feasibility Analysis

Abstract – In this chapter, the feasibility analysis of the problem is presented. It is based, mainly, on the issues and risks that were identified at an early stagy. However due to the substance of the problem, additional challenges were identified all the way till the end of the project. For this project, issues and risks are really depended, that is often due to the fact that in order to identify a risk you need to check what issue may cause a potential risk.

5.1 Issues

Since the very beginning of the project, many issues have arisen, and needed to take into account in order to proceed to the solution of the given problem. Issues can vary, because they can include different aspects, such as declaration of 'what is reliability of a software architecture,' or 'what is actually needed from the stakeholders'.

5.1.1. Software architecture evaluation based on reliability

The key aspect on the problem that needs solution is reliability. Moreover, what makes a software architecture reliable? Even more what is a reliable system when it comes to the lighting domain.

From the very start of the project, a concrete definition of the term reliability on a software architecture made in order to avoid conflicts late. To do so, brainstorm sessions took place, where different reliabilities issues proposed and finally conclude to one, the response time of the system. Based on that the evaluation should be made.

5.1.2. Philips lighting domain familiarization

In order to proceed and start working on a solution for the problem, one needs to be familiar with the context of Philips Lighting. Philips Lighting domain is huge and explicit. Usually, it takes employers quite some time to get familiar with the domain, the context, the tools and the way of working.

Moreover, for this project that was even more crucial. Since the goal of the project is architecture evaluation, we should be able to take an already implemented architecture of a Philips Lighting system, and evaluate it. It becomes clear, that familiarizing with the lighting domain is crucial to the goal of the project. The potential architecture candidates that will be evaluated are in lighting domain, and how a lighting system works is crucial for making a correct evaluation.

Communication and contacting the right people make the procedure of familiarizing with the context of Philips Lighting quicker.

5.1.3. Poor architecture documentation

The goal of the project is to provide architects with a tool that will make the decision procedure of implementing a new architecture easier. In order to do that, we need to see how an architect thinks. The 'place' to get such information is the documentation of already implemented architectures of lighting systems.

From the very beginning of the project, the research for documented software architectures, in order to understand the lighting domain, gave poor results. Either there was no documentation for projects or documentation was there but not as detailed as was expected and also not in sync with the actual implementation. Adding to that, the architects' choices were not described.

5.1.4. Communication with stakeholders

The project has to be in line with the architects, since they are the ones to use it. Getting requirements from each one, plus convince them to be part of the project was an imperative part of the project.

5.2 Risks

Potential risks that could have a large impact on the outcome of the project, identified at an early stage. However, risks were examined as the project continued. The most important risks and their strategies are mentioned below. Table 2, shows the potential risks, effects, and chance for each one to happened. Following, there is more explanation for each risk.

Table 2 – Risk Management Table

Risk	Effect	Chance
Pilot Project to follow	High	Medium
LM-IP Architecture	Medium	Medium
People involve	High	Medium
Define reliability	High	High
Weight reliability	High	High
Tool to use	High	High
LM-IP variables	Medium	Medium
No documented architectures	Medium	High
StarSense Project	Medium	High
Existing Architectures	High	Medium

5.2.1. Pilot project to follow

The intention at the beginning was to find a project inside Philips Lighting that was about to start, and follow the architect on the decision that he will make, and get clear view on how an architect works. By following a project called StarSense, we could have an insight on the software architects point of view, plus an architecture candidate to evaluate, since the architect would make one for that project.

However, although we started following the StarSense project, we had to drop that and find another project to follow, since there was the risk designing the architecture for that project may delay, causing also delays to our project. If that would happened, then the effect for the project would be not to have an architecture candidate as input to continue with evaluation process. Instead we focused on using the LM-IP projects architecture as an input candidate.

5.2.2. LM-IP architecture

In order to avoid the risk of not having an implemented architecture to evaluate, or an existing architecture to evaluate, we decided to follow the one that had the most documentation among all the architectures, the LM-IP architecture.

However, since a lot of information needs to be understood, mainly about how that system works, people that are actually participate in the LM-IP project, invited to participate in this project, to eliminate the risk of misleading results.

5.2.3. People involve

The system architecture evaluation project, required a lot on knowledge transfer. Architects and engineers that participate in the LM-IP project, are crucial to involve. Therefore, planning and meeting on a regular basis was needed in order not to lose valuable time.

5.2.4. Reliability of different components

The goal of the project is to evaluate software architecture based on reliability criteria. However, how to define reliability on architecture and moreover on each component, that is part of an architecture? At first, there was the issue to decide which aspect of reliability to use when evaluating an architecture.

5.2.5. Tool to use

During the research that made, methodologies and tools were found to be followed towards architecture evaluation. A risk here is selecting a tool that does not work as expected because of lack experience.

Another risk that a potential tool could have is not having the results that are needed. Probably a tool might found really promising, but in the end what it provides to us won't be the intended. A potential solution to that risk would be to be able to extend such a tool with new methods.

6.System Requirements

Abstract – In this chapter, the project's requirements are presented, which are the result of having meetings and discussion with the main stakeholders of the project, the architects. Then, the identified design competencies that are important for the success of this project are given, and finally the tool to continue our approach is presented to conclude this chapter.

6.1 Introduction

In chapter 4, there is the description of the problem analysis. The requirements of the proposed solution are analyzed in this section. First of all, stakeholders investigation was really important, because they are the ones that use the tool; moreover they are the ones to state the systems requirements.

In chapter 3, the stakeholder's analysis describes the whole procedure that defined the ones that are involved in the project. The rest on this chapter describes the systems requirements that the stakeholders want from the proposed solution to have.

Based on the architect's needs, the system's requirements can be categorized in 3 main categories:

- Model different parts of a use case scenario of an architecture
- Run simulation for the modelled scenario
- View simulation results

6.2 System requirements

The following section provides the gathered requirements from the architects. Previously, the categorization of the requirements mentioned; Table 3 follows this categorization and presents the listed requirements in such order.

Table 3 – System Requirements

ID	Requirement	Note	Priority
SR1	User Input	A user could be able to create different models that are defined in an architecture model.	Must
SR2	Data visualization	Having input from user, the output can be visualized, in different diagrams.	Must
SR3	Create components repository	A place to model the different parts (components) that participate in a specified use case.	Must
SR4	Modeling specification of components	A place to give the components behavior specifications, towards a use case.	Must
SR5	Create System model	Model a composite structure of a use case that is part of an architecture.	Must
SR6	Create different system models for same repository	Having specified a components repository, make several models of a composite structure to run simulations.	Should
SR7	Modelling Resources	A place to specify resource environment specifications.	Must
SR8	Modelling usage profile	A place to specify the usage characteristics of the use	Must

		case. What is triggering the system?	
SR9	Create allocation diagram	Allocation diagram maps components to resources.	Must
SR10	Validation of components repository	Check if an implemented repository has validation issues.	Should
SR11	Validation of System model	Check whether the composite structure has validation issues.	Should
SR12	Extend components repository	Ability to add/ remove components from the repository.	Should
SR13	Change/update specifications of components	Ability to change specifications on each component after modelling them.	Should
SR14	Change/update System model	Ability to change the composite structure after implementing it or after run a simulation.	Should
SR15	Change/update resources	Ability to add/ remove resources from the resource environment.	Should
SR16	Change/update usage profile	Ability to change/update the usage profile, by adding or removing things between different simulation runs.	Should
SR17	Define different configurations files.	Have different configuration files per simulation.	Must
SR18	Run simulation of the modelled environments, based on the configuration files	Choose configuration file to run the simulation.	Must
SR19	Visualize results of the simulation	Diagrams showing the data collecting from the simulations.	Must
SR20	Visualize and compare different simulation results	Diagrams showing results from different simulation runs.	Must
SR21	Export simulation data	Save simulation data to an excel file.	Should
SR22	Connect different repositories	Create system model by referring to different repositories of components.	Must
SR23	Create library of components	Create library to include different parts that are commonly used in system's model, for later use.	Should
SR24	Ease of use tool	Friendly and easy to use environment of tool.	Should
SR25	Compatible with Philips OS environment	No special modification to use the tool.	Must

6.3 Design opportunities

For this project three design criteria were selected by the very beginning, and in this section we are going to analyze them. Moreover, by having this criteria in the design, we show what our system will be able to do. At the end of the project, these criteria were reflected upon and explains how the quality of the design was improved for the important criteria.

The design criteria that are of interest for this project are:

- Flexibility
- Reusability
- Complexity

Flexibility implies a design that can easily accommodate the changes. This can either means adding more of the same kind of functionality or adding different functionality. Since we are in the Philips lighting and we need to evaluate architecture candidates, an example of that could be that we have a sensor that mainly detects motion. However, we implement another sensor component that does the same, but its functionality is different. In the end we will see which of the two sensors components is

more beneficial for us, but our system should be able to handle this changes, and produce results.

Reusability implies that designing a software system in such way that components of a developed system can be used again in developing new applications. Reusability is an important factor in producing low cost applications. Since the architecture evaluation is for lighting systems, it is really crucial that when implementing a system to evaluate the architecture candidate, to be able to use that again, or part of it such as the components that participate.

Complexity is the last of the three selected design criteria. The whole effort is to provide the architect with a procedure to make the choices he needs easily and not spending too much time in creating different models.

6.4 Selected tool for evaluation

This section provide information on tools for architecture evaluation, that found during the research that was made. Table 4 gives a description of each tool, with the advantages and disadvantages of each tool while Table 5 presents a comparison between the different tools that guide to the selection of one. 3

Table 4 – Selected tools for architecture evalaution

Tools	Pros	Cons	Description
SHEsim/POOSL	Analysis and realization of a system. Evaluation and performance analysis.	New tool maybe not support will provided. Standalone tool, connect it with e.g. Eclipse is not sure.	Uses POOSL Language (similar to UML notation) to model hardware/software systems. Build-in methods and techniques for evaluation.
Palladio Tool	Predicting reliability of a system by solving parameter dependencies, determining probabilities of system states, generate and solve Markov chains.		1) Make Palladio Component Model. 2) Make use case scenario and then define reliabilities/ dependencies etc. 3) Allows you to model different aspects of a system, using a domain specific modelling language
SBRA Technique	Uses framework in eclipse to generate code from model.	Not commonly used	1) Define scenarios for interaction between components 2) Model Component Dependency Graph 3) Run Algorithm to analyze reliability of each component
PSO Technique	Palladio component modelling + QML. Uses Ecore plugin to model inside eclipse and then define framework to evaluate reliability of a system.	QML, lack of knowledge.	1)Make Palladio component model 2)Model quality requirements with QML 3)Run algorithm to evaluate 4) Provide results in graphs

Table 5 – Comparison of selected tools

	SHEsim/POOSL	Palladio	SBRA Technique	PSO Technique
Stand Alone tool	✓	✓		
Open source		✓		
Eclipse Based		✓		✓
Generated from Ecore		✓		✓
Probabilistic methods used for measurements	✓	✓	✓	✓
Ready for use tool	✓	✓		
Documentation	✓	✓	✓	✓
Forum		✓		

Given the requirements from the stakeholders, and the description of the problem, the tool that is selected to continue the research for the solution, Palladio tool is selected for the purpose.

Palladio is an Eclipse based open source tool. Uses probabilistic methods to make the measurements, similar to the ones found in the research made, SBRA technique or PSO technique. As most of the other methodologies Palladio also needs a use case scenario of the systems architecture, with well-defined characteristics in order to be able to make measurements. Right now Palladio and Ecore, which is a plugin in eclipse similar to UML modeling, were the two tools that was actually a ready environment to work with. The difference is that Ecore is more general when Palladio is specific for architecture evaluation. A reason to follow Palladio than Ecore at first place was that Palladio is generated from Ecore.

As for the other methodologies, is that at this moment they were just step by step guidance to build a tool to use, although Palladio not only was a step by step methodology to evaluate architecture but also a ready-made tool to use.■

7.System Architecture

Abstract – In this chapter the overall architecture is described. Development process followed by the 4+1 model are explained, towards the tool that selected in chapter 6.

7.1 Introduction

In previous chapters, the descriptions of the problem, domain analysis and the requirements were presented. This chapter provides an overview of the system's architecture. First, this overview is given and then we go into more details about each of the parts of this architecture. The purpose is to give the whole image to the reader and later on explain what each part represents.

7.2 System's architecture overview

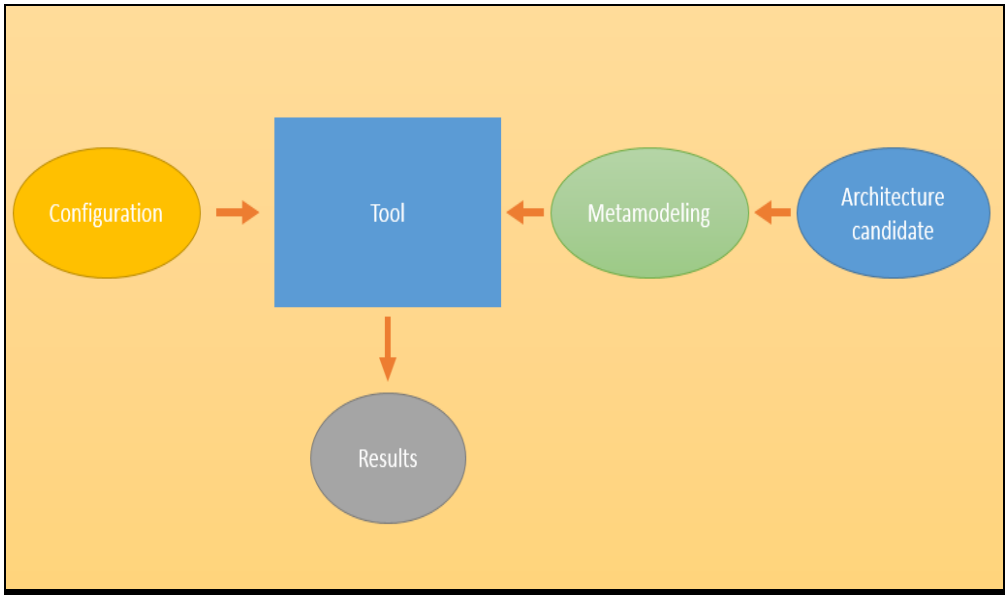

Figure 5 – System Architecture

Figure 4, shows the architecture overview of the system, indicating the inputs of the tool and the outputs it gives.

The system's architecture contains the following:

- An architecture candidate.
- Metamodeling of the candidate.
- Palladio tool.
- Configuration.
- Results.

7.3 Development process

The development process starts from the collection of the requirements (requirement phase). Later on, the software architect, gives the specification of the architecture and

the components (specification phase). When everything is specified, performance predictions can be carried out from the Palladio tool (QoS-Analysis). With the results the software architect can either continue to the implementation of the architecture (deployment phase), or change specifications to run another round of simulation (provisioning phase). Figure 6, shows the development process as it is described in this section.

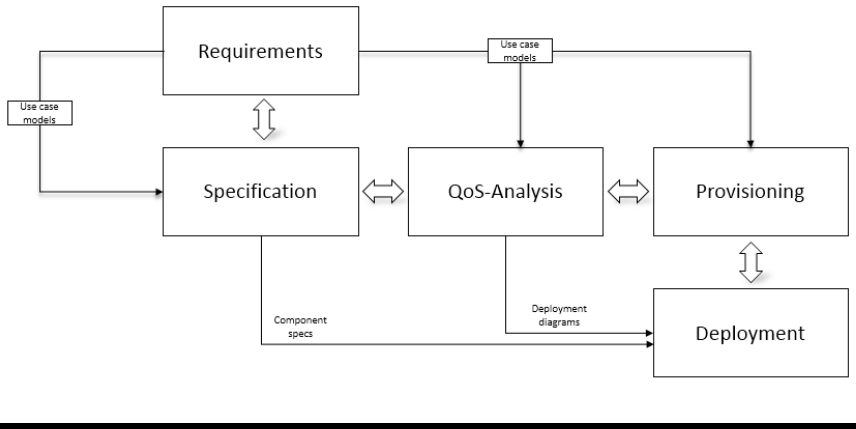

Figure 6 – Development Process

Given an architecture candidate, an architect wants to measure the performance as stated in the requirements section. In order to do so, different metamodeling techniques are needed. When we are referring to a software architecture model different layers of that architecture needs to be defined properly, to be given as inputs for the Palladio tool.

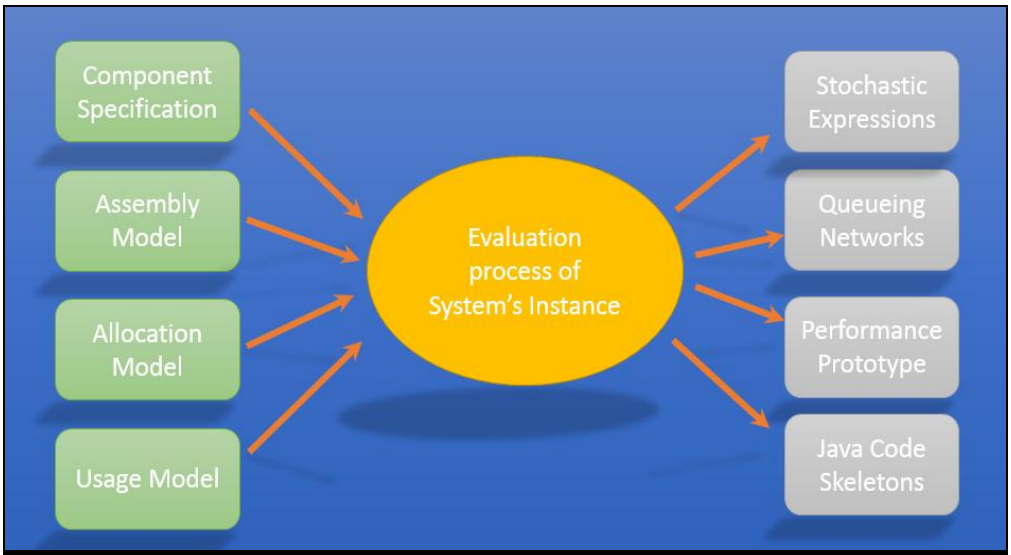

Figure 7 – Palladio Tool inputs

Palladio provides a domain-specific modelling language for each input, which is restricted to concepts known to this role. Figure 7 shows the inputs Palladio needs in order to evaluate a potential architecture. Let now have a clearer view of what is shown in Figure 7 as input:

- **Component Specification:** components taking part in an architecture candidate, and their performance-related behavior.
- **Assembly Model:** composite structure and communication between the components introduced.

- **Allocation Model:** model of the resource environment plus the allocation of the components to this environment.
- **Usage Model.**

After providing this modelling information to Palladio tool, and before running the simulation, a configuration is needed to determine the circumstances of the experiments. That is the final input for the tool. All the inputs combine and create an instance of the candidate architecture model. That instance is the one that will be evaluated.

Palladio offers different performance evaluation techniques. For analyzing use cases without concurrency, an instance can be transformed into a stochastic regular expression (SRE), which offers a fast way of predicting response times in presence of resource demands specified as general distribution functions. For cases with multiple users, an instance can be transformed into a queuing network based simulation model. The simulation model is less restricted than the SREs, but its execution is usually more time consuming than solving the SREs. Finally, there are transformations to derive Java code skeletons from an instance, to provide a starting point for implementing the modeled architecture.

7.4 4+1 Architectural view

The 4+1 architectural view describes the architecture of software systems, based on the use of multiple, concurrent views. In each view, different stakeholders are considered. In addition, a selection of use case scenarios describes the overall picture as 'plus one'. Hence the model contains 4+1 views:

- **The logical view:** This contains information about the various parts of the system. In UML the logical view is modelled using Class, Object, State machine and Interaction diagrams (e.g. Sequence diagrams.
- **Process view:** The process view considers non-functional, dynamic aspects such as performance, scalability and throughput. It addresses the issues of concurrency, distribution and fault tolerance. The process view can be represented with sequence diagrams, activity diagrams and communication diagrams.
- **The development view:** The development view focusses on software modules and subsystems. In UML, Package and Component diagrams are used to model the development view.
- **The physical view:** The physical view encompasses the nodes that form the system's hardware topology on which the system executes. It focusses on distribution, communication and provisioning and is represented by deployment diagrams.
- **The use case view:** This view describes the functionality of the system from the perspective from outside world. It contains diagrams describing what the system is supposed to do from a black box perspective. This view typically contains Use Case diagrams. All other views use this view to guide them.

These five views are connected to each other, Figure 8. The logical view will help the developer to extract the development and process views. Then these two view can be used together to come up with a physical view. During this process the scenarios can be used to see the big picture and make all these view consistent with each other.

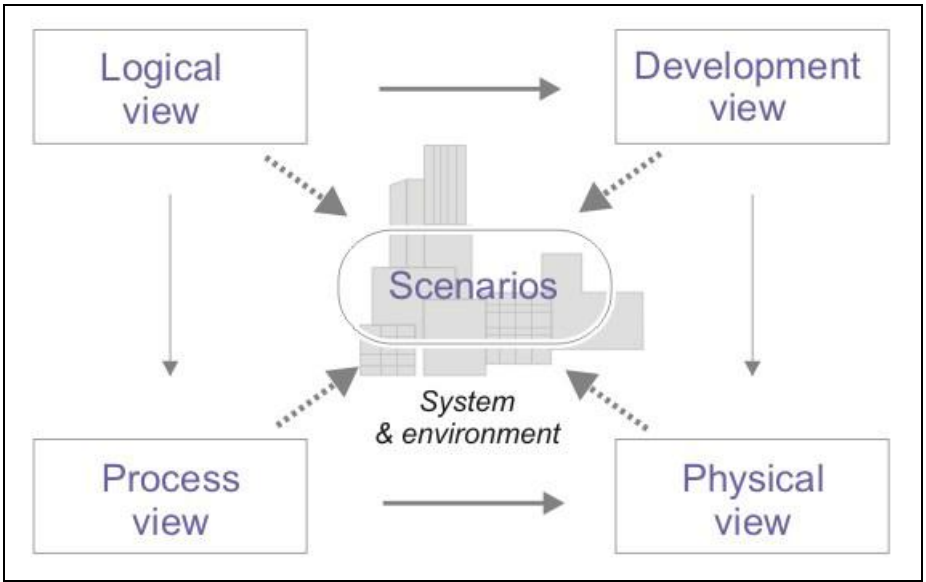

Figure 8 – 4+1 Architectural View

7.4.1. Use case scenarios

Use case scenarios show how users interact with the system. Basically, it shows the behavior of the system as seen by its end users and other stakeholders. This behavior needs to reflect the requirements stated in the domain analysis description. In this section, the five main use case scenarios are explained. Table 6 - Table 9 describe the use case scenarios that are implemented by the system.

Table 6 – Provide Models to the system Use Case

Features	Description
Use Case:	SAE1
Name:	Provide Models to system
Scope:	Software Architecture Evaluation (SAE)
Level:	User-goal
Primary Actor:	System or Software architect
Stakeholders & Interests:	Project Manager and QPL (Quality Project Leader)
Precondition:	Tool is installed and running Software architect (SA) have concrete specification of what is going to be modelled
Minimal Guarantees:	t.b.w
Success Guarantees:	Successfully provide the 5 types of modelling of a software architecture.
Main Success Scenario:	(1) SAE: Provides environment so SA can model a system (2) SA: Models components used in his scenario, retrieves existing components and creates new one if needed (3) SA: Gives the service effects specifications (SEFFs) (4) SA: Makes composite structure (5) SA: Gives the resource envi-

	ronment variables (6) SA: Makes usage profile (7) SAE: Runs simulation (8) SAE: Provides results

Table 7 – Differentiate Structure Use case

Features	Description
Use Case:	SAE2
Name:	Differentiate composite structure
Scope:	Software Architecture Evaluation (SAE)
Level:	User-goal
Primary Actor:	System or Software architect (SA)
Stakeholders & Interests:	Project Manager and QPL (Quality Project Leader)
Precondition:	SA had already model and run once a simulation
Minimal Guarantees:	t.b.w
Success Guarantees:	Successfully change composite structure
Main Success Scenario:	(1) SAE: Provides the SA with structure that was already implemented (2) SA: Changes the structure of the communication and the interaction between the components that are used in the scenario (3) SAE: Runs simulation (4) SAE: Provides results

Table 8 – Differentiate Triggering events Use case

Features	Description
Use Case:	SAE3
Name:	Differentiate events triggering usage scenario
Scope:	Software Architecture Evaluation (SAE)
Level:	User-goal
Primary Actor:	System or Software architect (SA)
Stakeholders & Interests:	Project Manager and QPL (Quality Project Leader)
Precondition:	SA had already model and run once a simulation
Minimal Guarantees:	t.b.w
Success Guarantees:	Successfully run different scenarios with triggering events
Main Success Scenario:	(1) SAE: Provides the SA with the modelling architecture that was made (2) SA: Change the usage scenario(e.g. number of events) (3) SAE: Runs simulation (4) SAE: Provides results (5) SA: change again usage scenario (6) SAE: Runs simulation

	(7) SAE: Provides results (8) SAE: Compare results in same charts

Table 9 – Differentiate on Non-functional requirements

Features	Description
Use Case:	SAE4
Name:	Design same components, differentiate them in non-functional requirements
Scope:	Software Architecture Evaluation (SAE)
Level:	User-goal
Primary Actor:	System or Software architect (SA)
Stakeholders & Interests:	Project Manager and QPL (Quality Project Leader)
Precondition:	SA had already model and run once a simulation
Minimal Guarantees:	t.b.w
Success Guarantees:	Successfully run different scenarios with triggering events
Main Success Scenario:	(1) SAE: Provides environment so SA can model a system (2) SA: Models components used in his scenario (3) SA: Gives the service effects specifications (SEFFs) (4) SA: Makes composite structure (5) SA: Gives the resource environment variables (6) SA: Makes usage profile (7) SAE: Runs simulation (8) SAE: Provides results

Table 10 – Comparison of results, to PUGH Sheet

Features	Description
Use Case:	SAE5
Name:	Comparison of Results
Scope:	Software Architecture Evaluation (SAE)
Level:	User-goal
Primary Actor:	System or Software architect, Quality Project Leader
Stakeholders & Interests:	Project Manager and QPL (Quality Project Leader)
Precondition:	SA had already model and run simulations
Minimal Guarantees:	t.b.w
Success Guarantees:	Successfully run different scenarios with triggering events
Main Success Scenario:	(1) SAE: Provides simulation results (2) QPL: Select to export results to excel sheet

	(3) SAE: Export results in excel files.

7.4.2. Logical view

In this view, the logical layout of the project is given. Mainly it shows the interfaces and the classes that participate in a use case scenario that is part of an architecture candidate that is going to be evaluated. For this project this view contains the components repository (interfaces and classes) as shown in Figure 9.

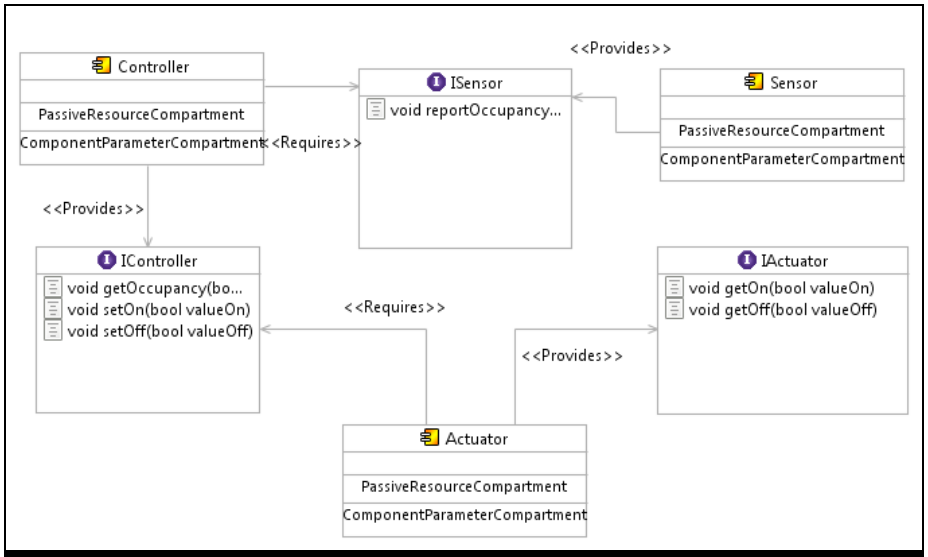

Figure 9 – Classes and interfaces on the components logical view

7.4.3. Process view

The process view describes the processes of elements that interact with each other. The parts (interfaces, classes) that introduces in the Logical View, have specification. These specification are analyzed in the process view with activity like diagrams. Adding to that the usage models are also behavior descriptive.

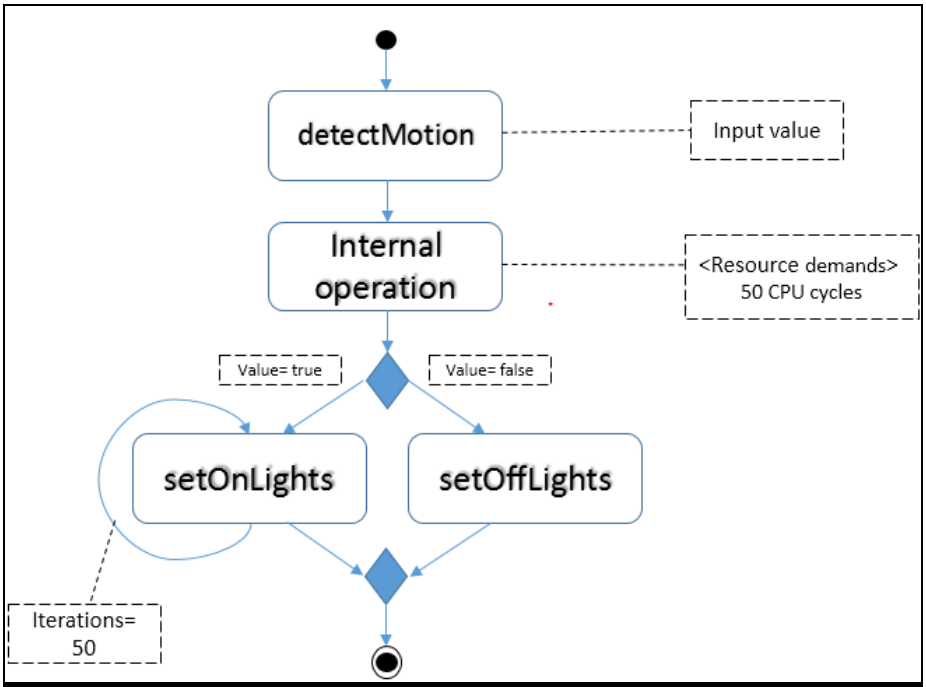

Figure 10 – Activity diagram showing specfification of component.

7.4.4. Development view

The development view shows the components, which will be the units of deployment. Since in the logical view there are the parts that take place in an architecture, the process view describes the behavior of each part, in the development view basically we can see the actual composite structure of the use case scenario. The assembly model shows the composite structure, and more over the communication between the different components which is achieved by 'provide' or 'require' an interface. Each interface contains methods that components use to communicate with the different parts inside a system.

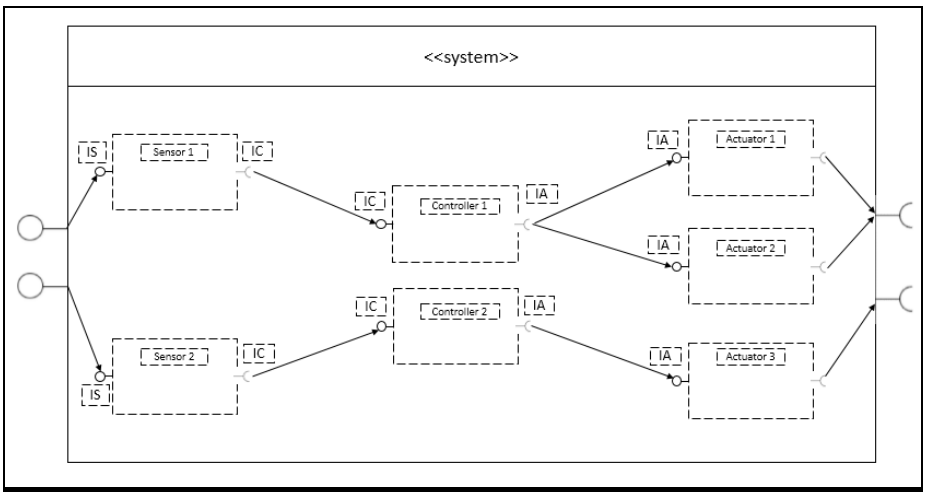

Figure 11 –Assembly model on the components process view

7.4.5. Physical view

The view described in this section deals with the deployment of the developed software on the hardware / execution environment. Figure 12, shows that, where we see the mapping on the resource containers, for each assembly context.

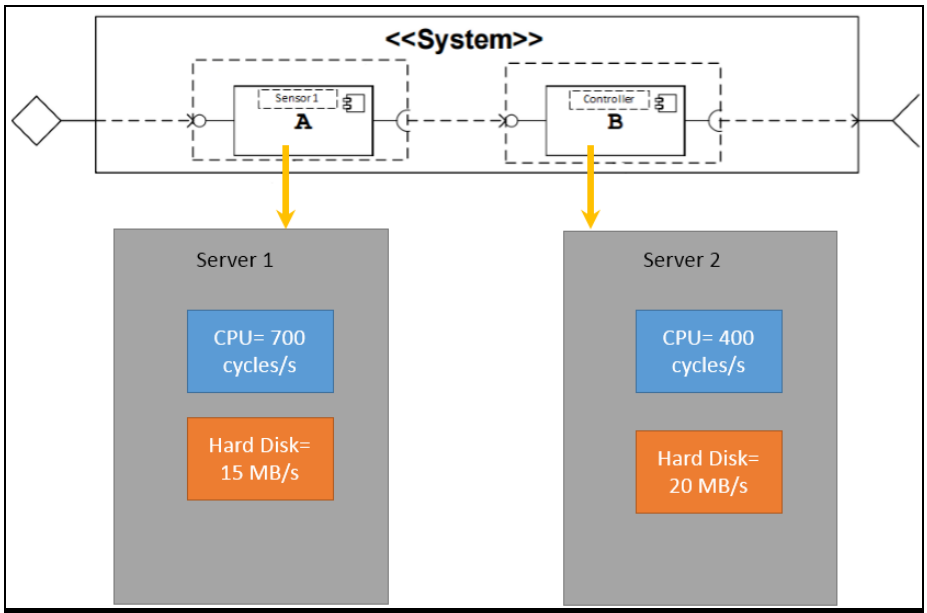

Figure 12 – Physical View.

8.System Design

Abstract – After explaining the main system architecture in chapter 7, this chapter provides a more detailed look on parts of the architecture. The metamodeling part of the inputs as long as the configuration are examined in this chapter. Core of the deign gives the main idea behind which the evaluation process is done.

8.1 Core of the design

The system architecture that described in the previous chapter, is the core where the design of the system is based. In Philips lighting, when implementing a new system the components that participate are mostly sensors, controllers, and actuators. When evaluating an architecture candidate, the need is to see which candidate performs better than the others. These can have different aspects, in implementation, composite structure, functionalities etc.

Following is the system design. It consists of the sensors, controllers, and actuators; however, there is another component, the multiplexer. A simple use case of a lighting system is a detection from sensor, reported to controller, and actuators take command from the controller according to that detection. The proposed multiplexer component helps us in the measurements when a controller has multiple inputs from different sensors. In a specified area several sensors exist to detect motion, and report that to a controller. In order to make our evaluations more concrete, we need to have a multiplexer component to get the inputs from the sensors.

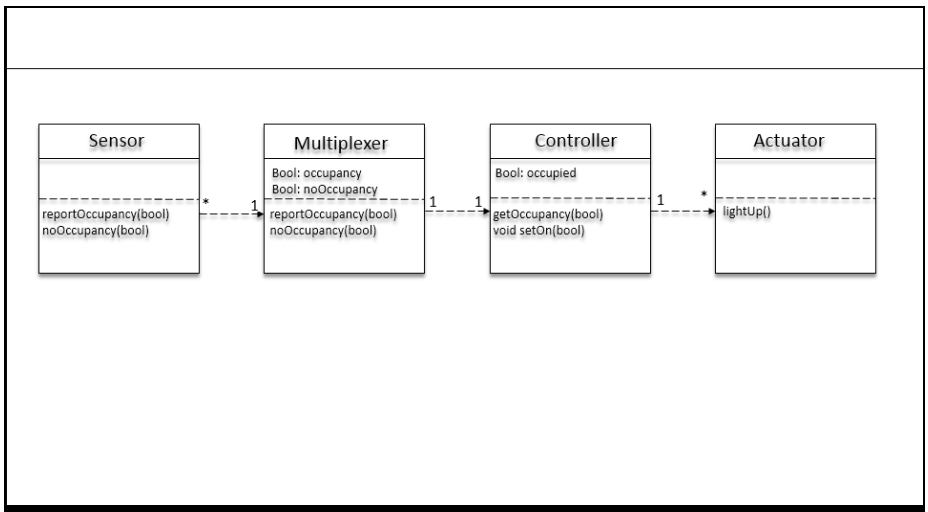

Figure 13 – Core of the Design

What is actually happening is that, when there is detection of movement by the sensors, we need to report that to the controller. The multiplexer does that. Its output is a logical OR, since it takes the inputs from the sensors, and send it to the controller.

However, when using the Palladio tool, there are several steps that need to be done before the composite structure. These steps are also part of the design, since they need to be in a concrete form, so that the simulation can provide results for the evaluation of an architecture candidate. The following sections describe that steps and design inside the Palladio tool.

8.2 Component modeling

The component modeling includes all that software parts that are defined in a use case scenario. More specifically, the components and each behavior. Firstly, all components need to be stored in a repository and then the behavior needs to be modelled, so later on, can be used for the deployment of the system.

8.2.1. Components repository

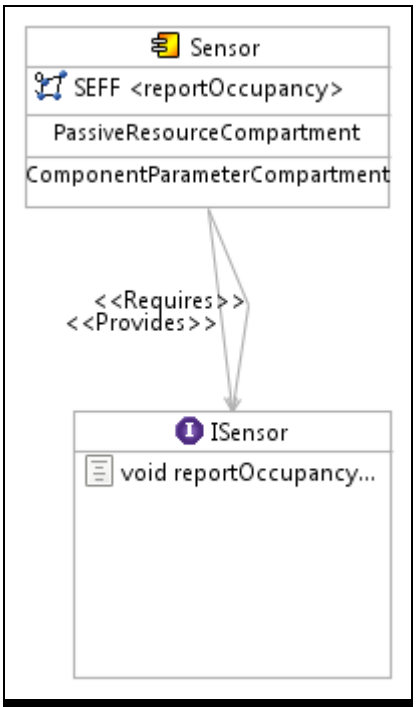

Figure 14 – Components repository.

Figure 14 shows a small example of a component repository. In such repositories, interfaces, data types, components can be declared. Components can provide or require interfaces. If a component provides an interface, this means that the component includes an implementation of the methods (services) declared in the interface. In the example of Figure 14, the Sensor component, provides the ISensor interface, and additionally, includes the implementation of the method that the interface provides. If a component requires an interface, can access the methods in the interface though the component that provide the interface. The next section analyses the specification of the services.

8.2.2. Service effect specification

After modelling the components repository, and additionally the services that each component provides or require from interfaces, the next step is to specify these services.

The service effects specification can be declared in a similar way as the activity diagrams do in UML. Main use, is to specify the performance of the components. What can be seen in this specification can vary. Control flow of actions, resource demands, and parametric dependencies.

Control flow shows internal actions, the component executes internally some code, or external call actions, which are the actions that a component requires from an interface.

Resource demands can be specified here. Components in order to execute their services use resources (CPU, HDD). Figure 15, shows an example of a service effect specification. CPU demands specified first, and then and external call action is specified. Parametric dependencies can cause different performance behavior. It is very common, to specify the resources demands depending on an input value. So for example if the resources needed are depending in the byte size of an input file, then the performance is different for files with different byte sizes.

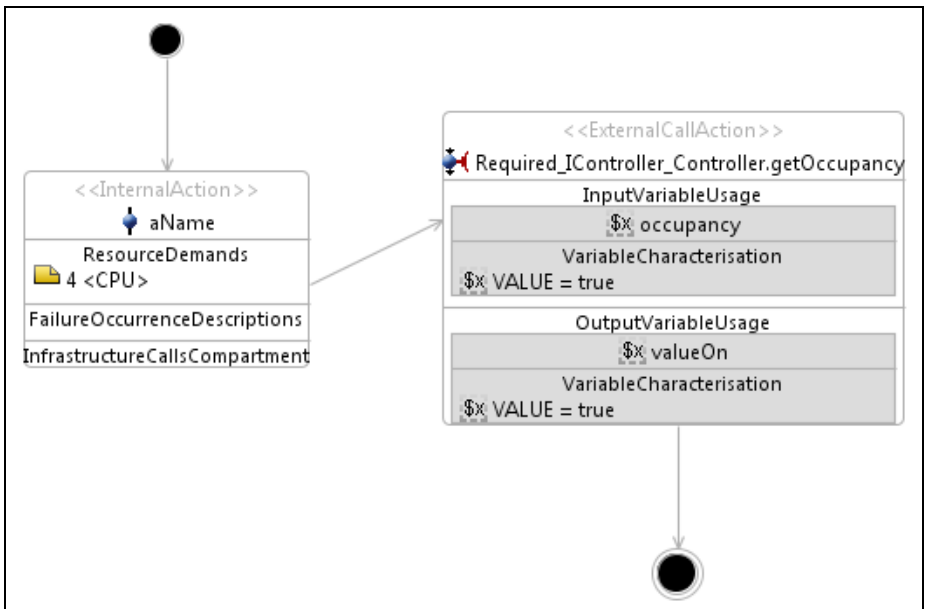

Figure 15 – Service Effect Specification.

8.3 Assembly model

After the components repository is imported, plus the specification of services have been declared, then the composition of the architecture can been deployed. The composite structure, represents the system that the architect needs to implement. Basically, the architect uses the components from the repository and deploys the system. At this point, no more specifications are needed, because everything is already implemented on the previous step.

A very important thing for this part of modelling is that for every component several instances may be used. In order to do so, all components used are in assembly context which represents an instance of the component.

The provided and required roles, first declared in the components repository implementation, have a crucial role here, since they help to connect the assembly components in the composite structure.

A small example of an assembly model shows Figure 16. It is clear that there are two instances of components, one for the sensor component, and one for the controller component. The provided and required roles are also obvious from the model. For example, controller requires a role from the sensor, as an input. If there were more instances of a sensor here, then also in the controller instance would be as many required roles from the sensor as the actual instances of the sensors.

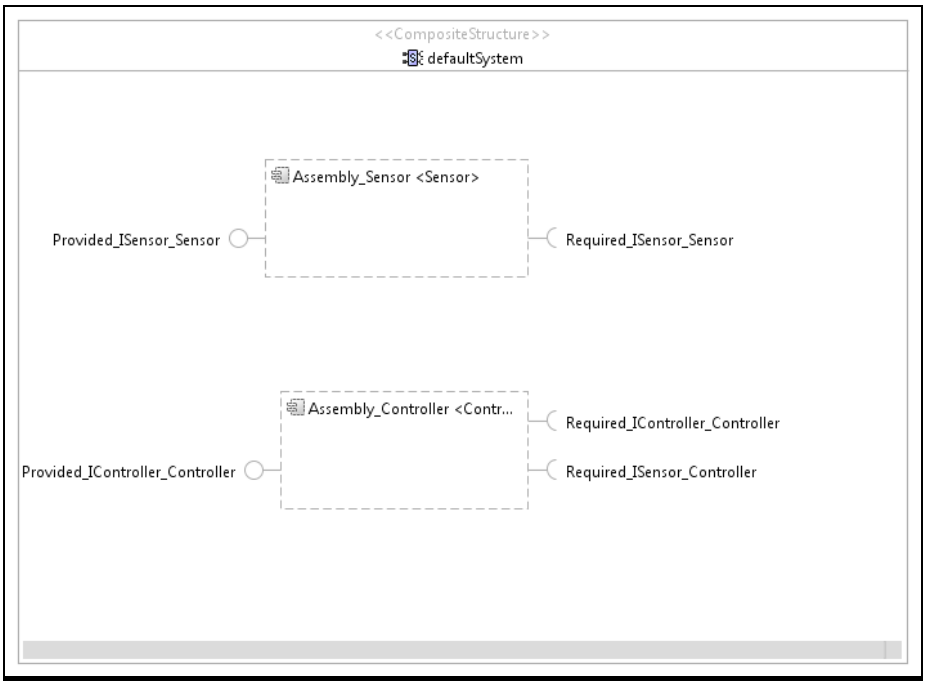

Figure 16 – Composite Structure/ Assembly System

8.4 Allocation model

There are two parts of modeling, when it comes to the allocation model. First, is the need to specify the resource environment, and second, to allocate the assembly components to the resources.

For the resource environment, resource containers can be specified (e.g. Server). The latter can contain CPU, hard disk, and connection delays. The resources have a processing rate, which is used to convert demands of the services specification into timing values. In section 8.1, the service effect specification explained, naming that the resources needed for each action should be declared. At the resource environment the whole processing rate of the resources is given, so when later simulation run, these two declaration will actually provide the numerical values of the simulation.

After specifying the resources, allocation context is used to specify that a resource container executes an assembly context. For example, having two servers helps putting different parts of a system in one server and others in the other server.

Figure 17, that follows shows exactly one Server, containing allocation context for every assembly context. For the example, all executions are done in one resource container.

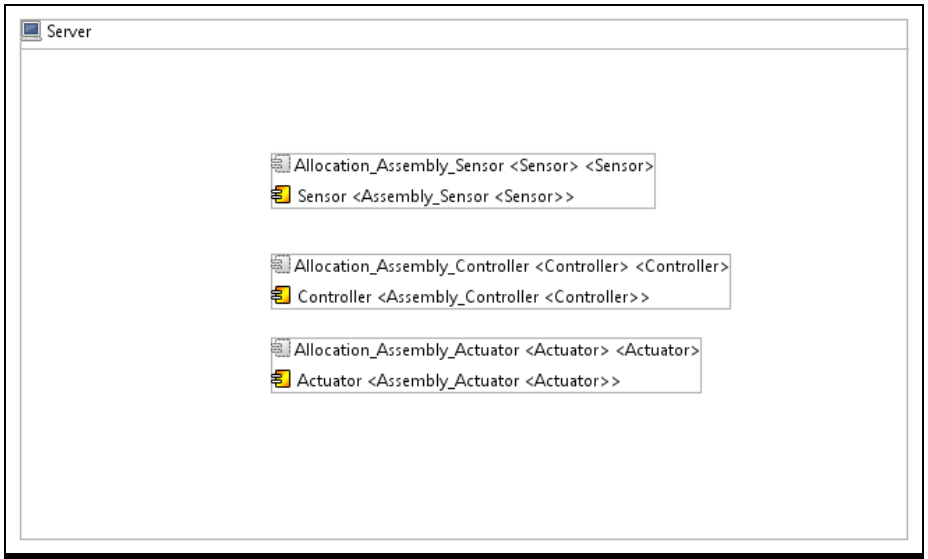

Figure 17 – Allocation Model

8.5 Usage model

In the usage model, what is actually shown is the user interaction with the system. It is the input that will trigger the system to be executed. It can have one simple input, or more complicated inputs, characterized by probabilistic actions. Control flows connected with branch actions, can show the user behavior. The more specified a usage model is, the more realistic the simulation could be.

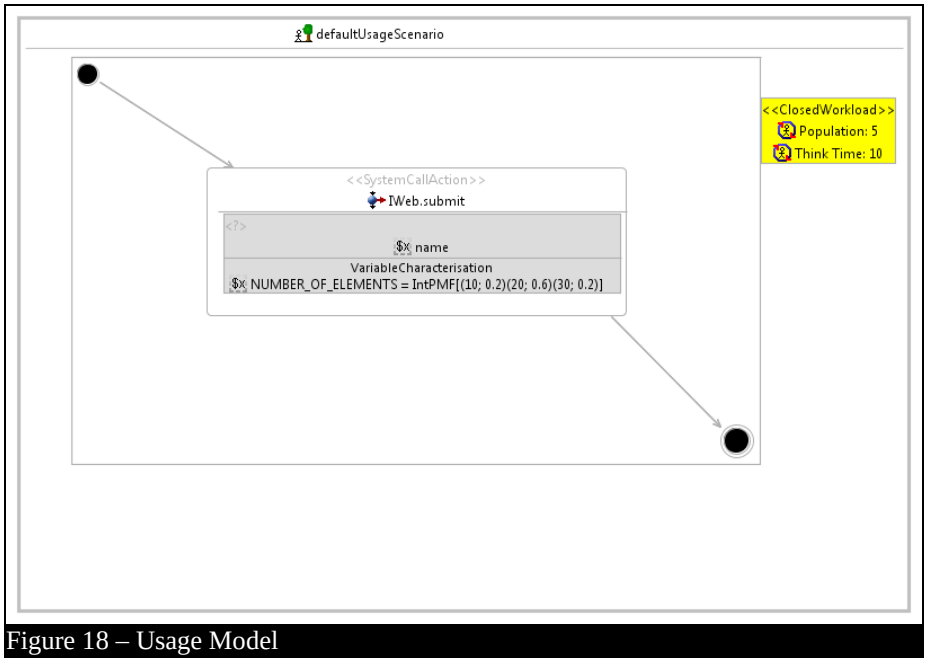

Figure 18 – Usage Model

Figure 18, shows a simple usage model, where the triggering event, can actually have different population of elements, since it is specifies as a distribution function of an integer. For example, this can help simulate a system where every minute we want to see the how many users are entering.

That is the final input in order to evaluate an architecture of a system. The only thing left is the configuration, before starting the simulation.

8.6 Configuration

At this point, the modeling part should have finished, and the simulation part is about to start. However, there is mapping missing of the part that will part in the simulation run. This mapping is the configuration file, which specifies the assembly model, allocation model, and usage model that are in the simulation. Besides that, the number of experiments is also declared at this point. Different configuration, can lead to different measurements, which may needed from an architect, in order to make his decision on the architecture to be implemented.

8.7 Transaction

An important factor for evaluating a lighting system's architecture candidate, is the transaction. Transaction, mainly should include information about network communication, and communication between the different components of a scenario. An example of that could be how many messages are exchanged between a sensor and a controller per minute.

Due to time limitation and no data information on this subject, we left it out of the scope when experimenting. Nevertheless, it should be included on future work, especially when network aspects need to be included.

■

9.Verification & Validation

Abstract – In this chapter, the capability of the proposed methodology and tool to provide a feasible result is shown. In order to do so, a use case scenario from an architecture candidate is selected. Afterwards, all the modelling behavior implemented as described in chapter 8. Finally, results are provided.

9.1 Use case scenario

To illustrate the performance of the selected tool, and the inputs that it needs, this section provides a use case scenario, of an implemented architecture inside Philips Lighting, the LM-IP. The system under analysis controls the behavior of light inside a building. The following example is mainly focuses on one scenario of the system.

9.1.1. LM-IP scenario

This LM-IP architecture, has many and different use cases. For the purpose of the verification and validation of this project, a specific scenario is selected. The Table 11 that follows presents that scenario.

Table 11 - Default Behavior of an un-commissioned system (Use case from LM-IP)

Features	Description
Name:	Everyday use
Description:	Default Behavior of an un-commissioned system program to react as single group of actuators.
Trigger event:	Actor entering a room. Every actuator has a sensor
Actors:	Office Workers, Installer, Worker
User Objective:	Performing tasks in appropriate lighting conditions
Pre-conditions:	1) All system components installed and powered 2) Power and network available
Post-conditions:	If any sensor detects movement all actuators turn on. If no movement detected, all actuators turn off.
Result:	Appropriate lighting conditions
Main Scenario:	1) Actor enters any room or area in the building 2) All lights in the sub-network are switched on to the defined light output 3) Option 1 If no occupancy is detected, the

	Actuator switches off If occupancy is detected by those sensors, entire Sub-net will switch on to the defined light level
Qualities:	One Group, One Light Level, Reaction Time 1s.

A small description of the scenario is the following: upon detection of motion, the sensor reports occupancy to the controller using the occupancy attribute. The controller reacts by turning on all actuators.

In case stable occupancy is present (the user moves under the actuator), the actuator reports this and the controller dims the lights to task level. In case not stable presence is detected, the controller decides to turn off the actuators.

For this scenario the components that take place are the following:

- Sensor
- Controller
- Actuator

In order to our approach more feasible, we introduce a new component, a Multiplexer. The purpose of that was to make more concrete our measurements. This components takes as input multiple events from the different sensors. Moreover, when simulation is done it is able to provide results and measurements for each sensor, and each method that a sensor contains. This is analyzed more in section 9.3.

For the rest of this chapter, this is the requirements specified for the model that are implemented in the next sections.

9.2 Input models/ validation

As described in chapter 8, the designing part of the input model is crucial since they need to be in specific forms in order to be used for simulation. Taking into account the components included in the scenario and their behavior, the repository and the specification of them can be constructed.

9.2.1. Components repository

Based on the description of the use case, Figure 19, presents the components repository. The design of it follows the specification named in chapter 8. The components that participate in the scenario are implemented. Roles of providing and requiring an interface are also there. The only thing that differentiates is the Load Balancer component, whose implementation is for handling requests from different sensors.

The service of each component, is also declared in the repository. The main purpose of the use case is to report the occupancy, if a sensor detects it. Different components implement different services. For example, Sensor reports occupancy, while the Controller requires that report in order to send to Actuators message whether to turn on or off.

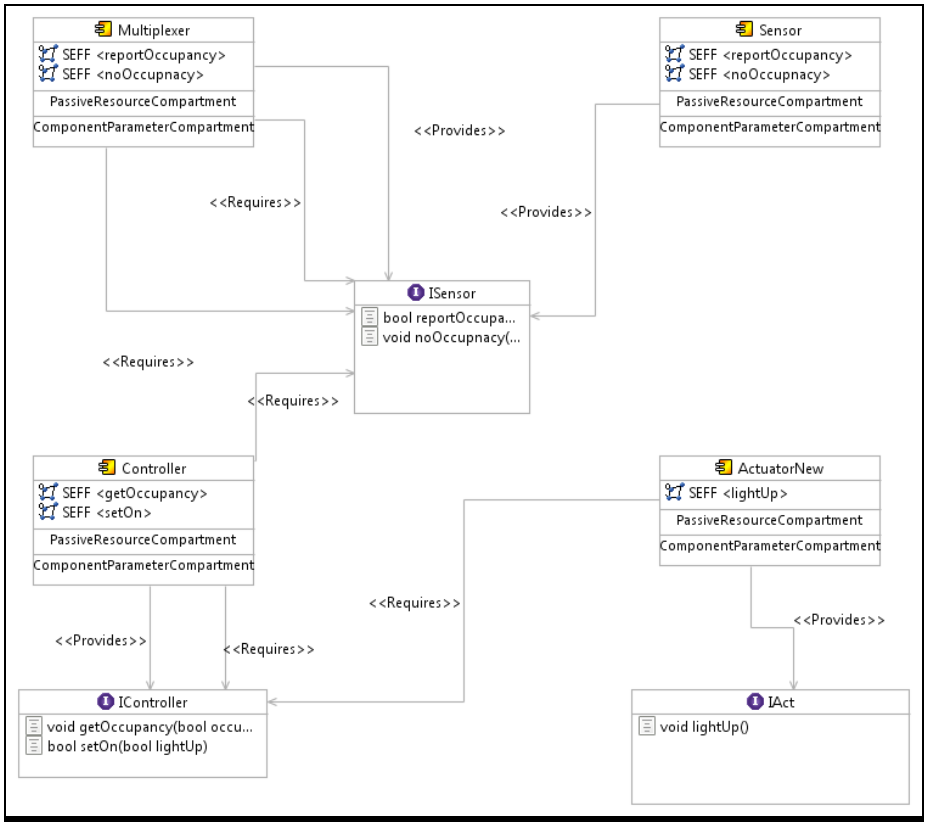

Figure 19 – Components repository for specific scenario

Following the design of the repository is to provide the specification of each components services. From Figure 19, it can be observed, which component provides which interface, and which component requires an interface.

Below, Figure 20 and Figure 21, show the implementation of the reportOccupancy and noOccupancy services. Both contain internal and external call actions. In the internal action the CPU load for the execution is declared. Both Sensor and Multiplexer components using them. However, there is difference between the uses. Sensor just detects or not detects motion. According to that, the Multiplexer, implements the behavior of each service.

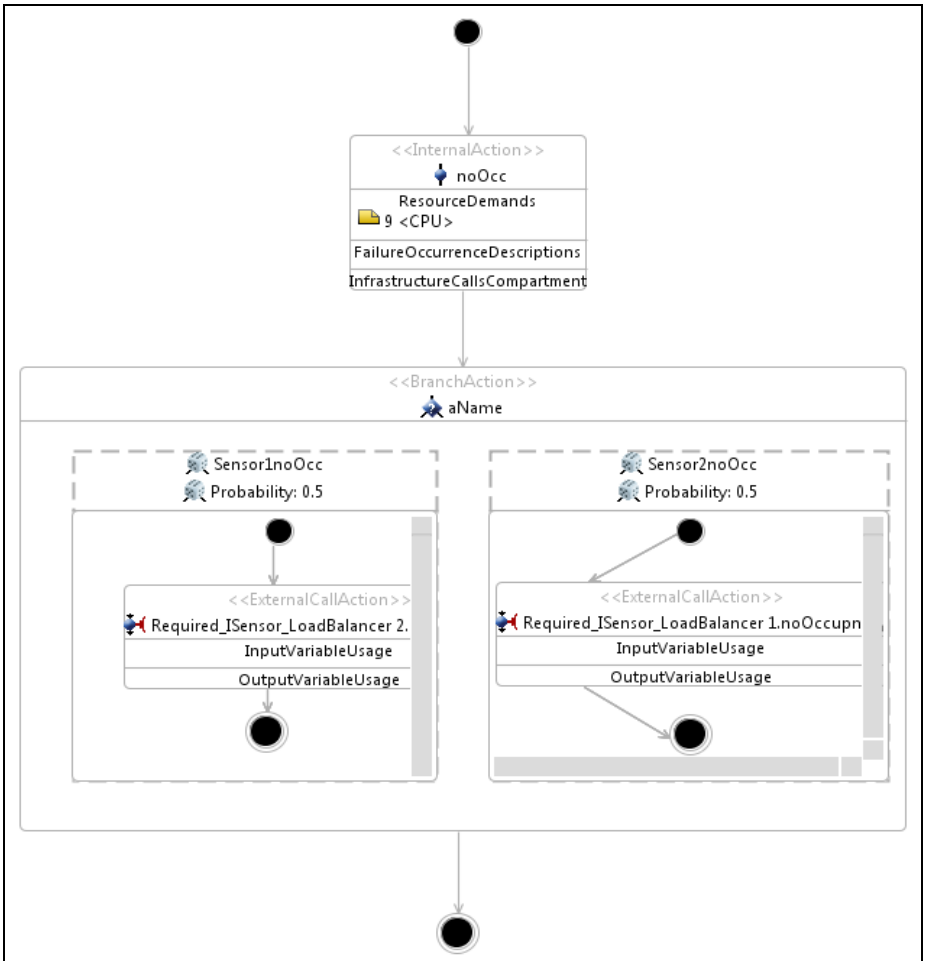

Figure 20 – noOccupncy service in Multiplexer

For the Multiplexer, the implementation is similar, thought the branch actions and probabilities refer to the sensor that will be triggered for the input event.

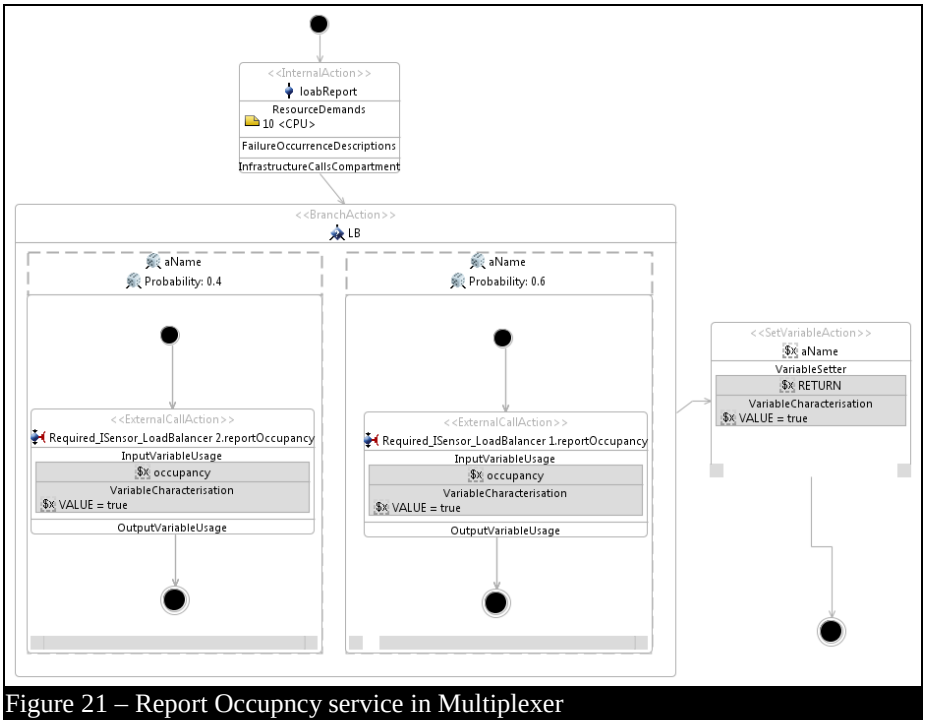

Figure 21 – Report Occupncy service in Multiplexer

The controller component need to get the report of the occupancy. And after it gets the occupancy or not, can provide the actuators with a turn on or off message. The service the controller provide is the get Occupancy. Figure 22 shows the implementation of that service. Again, the amount of CPU load is declared first. The external call action here is to get the output of the reportOccupancy service.

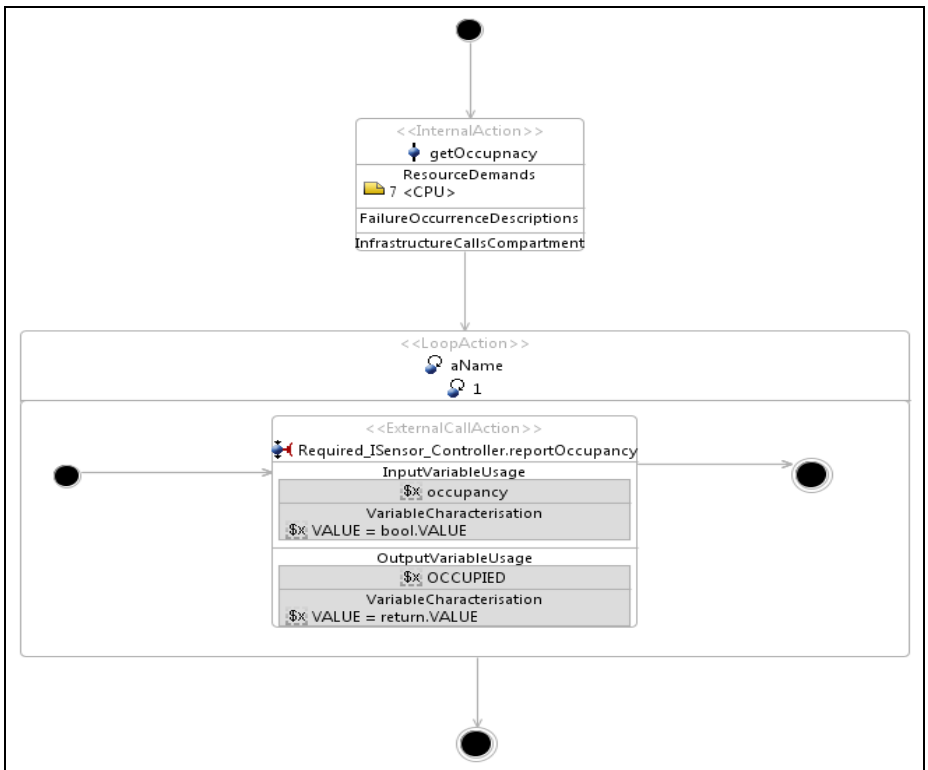

Figure 22 – GetOccupancy implementation in Controller

9.2.2. Assembly model

Once the components taking part in the use case are imported and the services have been declared, the Assembly model can be composite, as it is describes in chapter 8. For the scenario that is examined, the composition of the components shown in Figure 23. The flow between the different parts is clear and reflects the specifications of the scenario description. First, sensors detect the occupancy, then report that to the controller and finally the information arrives at the actuator.

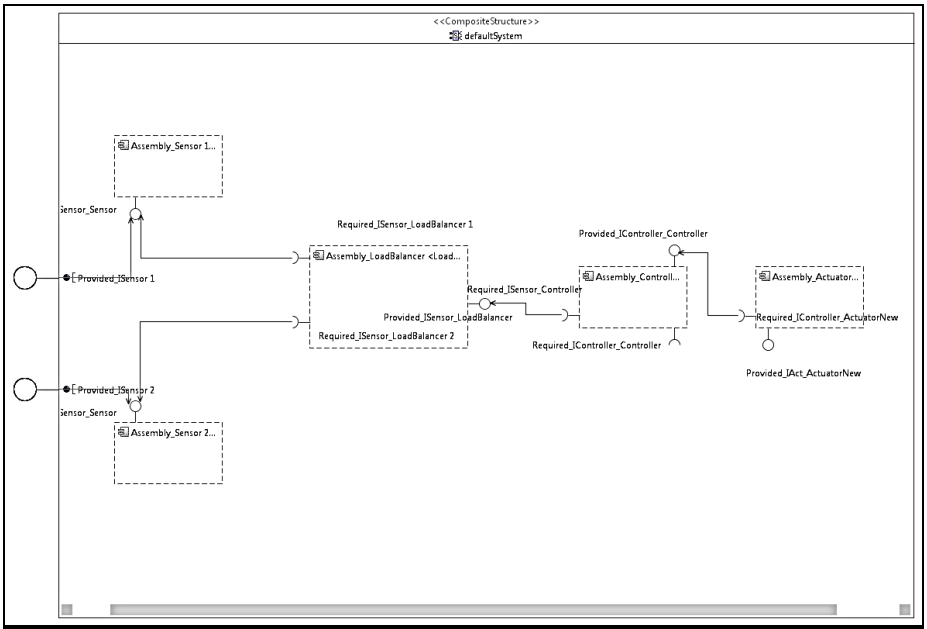

Figure 23 – Composite structure of the system

9.2.3. Allocation model

For the allocation model, first, the resources need is declared. Only CPU loads are decided to be the performance measurement. In addition to that, the whole system is running on a server, which can operate 700 CPU operations per second.

9.2.4. Usage model

For the usage model, we defined two scenarios containing branch probabilities each one. Since we modeled two sensors then, we need to declare these two scenarios each one affect one of the sensor. However, in the end both scenarios are taken into account or the simulations. Each scenario contains probabilities that either the sensor will detect motion or not. Moreover, for that specific usage model, ten event are modelled to trigger the system repeatedly, with a time difference of 1 second between each other, for Sensor 1, and for Sensor 2 only 5 events are modelled. Considering an office room, the placing of sensors gives different distribution of triggering event. Figure 24, shows the Usage model.

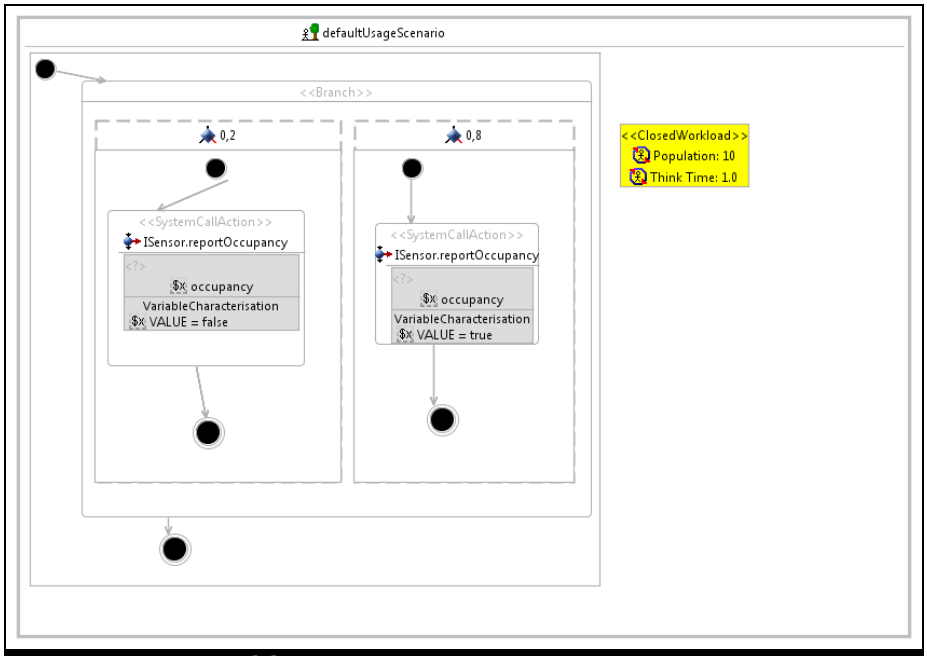

Figure 24 – Usage Model

9.3 Results and verification

This section provides the simulation results for the inputs provided to the Palladio tool. For the inputs that were given to the tool, we can have simulation results on the response time of the system, on response time of the different methods inside the 2 sensors, plus utilization of the resource container. This section is going to describe these results, and draw some conclusions about the experiments that can be made in such a tool. For the purpose of our experiments, mainly the stochastic process algebra and queuing networks simulation were used.

9.3.1. Response time of the usage scenario

In section 9.2, the inputs are given, more specifically components take place, behavior of them, assembly model, allocation model, and usage model. Simulation results show performance of the system provided. Performance results for the whole system; however results for specific components can be also made. Results can show from how much time one event consumes from the time it detected from sensor until it affect the actuator until which events trigger the whole system during the simulation run.

In order to make it more understandable results in graphs follows. These graphs are actually the outcome of the simulations. For the usage scenario, we can have results such as execution time per event, meaning how much time the system will run starting from a triggering event, distribution of the events triggering the system in the execution time (Figure 25), shows when events are coming. Then probability of reaching a response time according to specifications defined to the system (Figure 26), and finally the cumulative distribution function of the usage scenario (Figure 27). The cumulative distribution function shows the time needed for the coming event in added probabilities. The time needed for one detection from the sensor until the actuator turn on, was 0,177 seconds. However, we can have results also per method, basically because of the use of the multiplexer. For our experiment the methods used by the two sensors were the same so we have similar results in time

measurement. Still that is great input to know as a simulation result, which method and from which sensor was triggered, and moreover to see the time that it consumes.

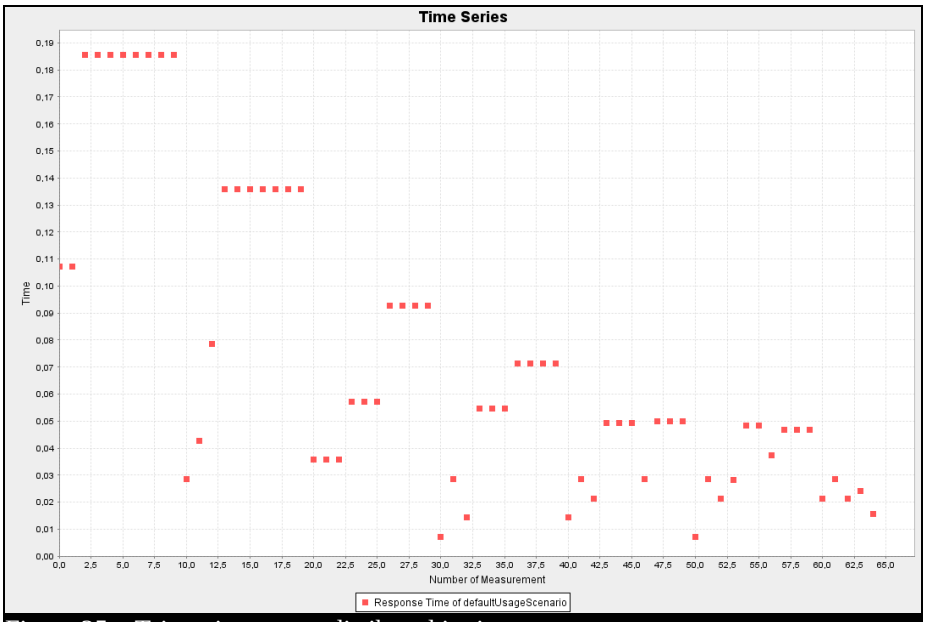

Figure 25 – Triggering events distibuted in time.

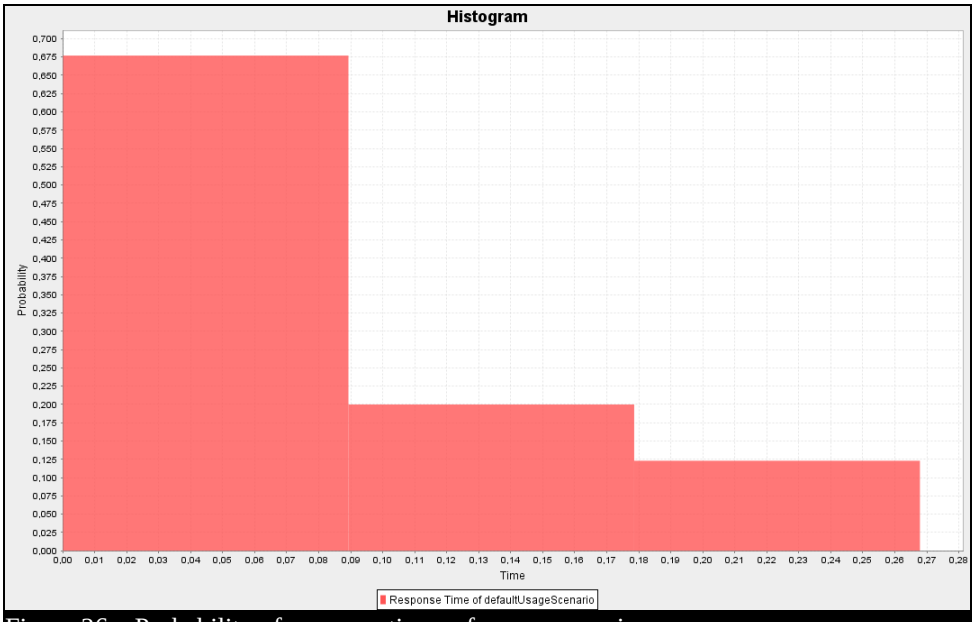

Figure 26 – Probability of response times of usage scenario.

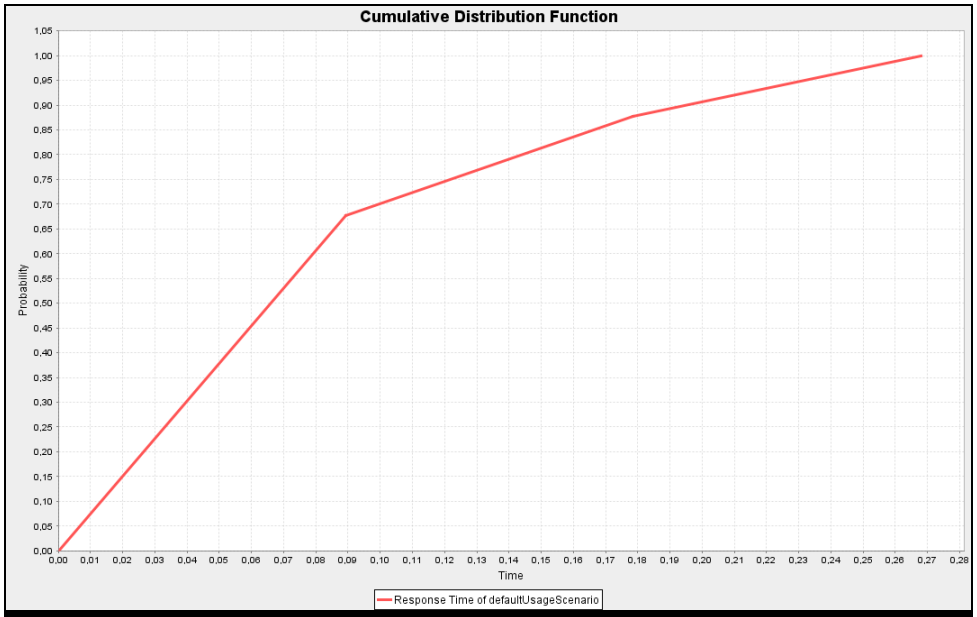

Figure 27 – Cumulative distribution function of usage scenario.

To get more information on the performance results, by introducing the multiplexer component we can have more clear view on the events inside the systems. Above we presented the results for the usage scenario, now we will show the same results with the difference that now we will show which method and from which sensor now triggers the system. For this experiment we introduced two sensors that detect occupancy or not. If we now see the triggering events coming to the system, we can actually see which kind of event is triggering the system in Figure 28. Basically that figure shows, whether the event that comes trigger sensor 1 or sensor 2, and whether it is an occupancy detection or not.

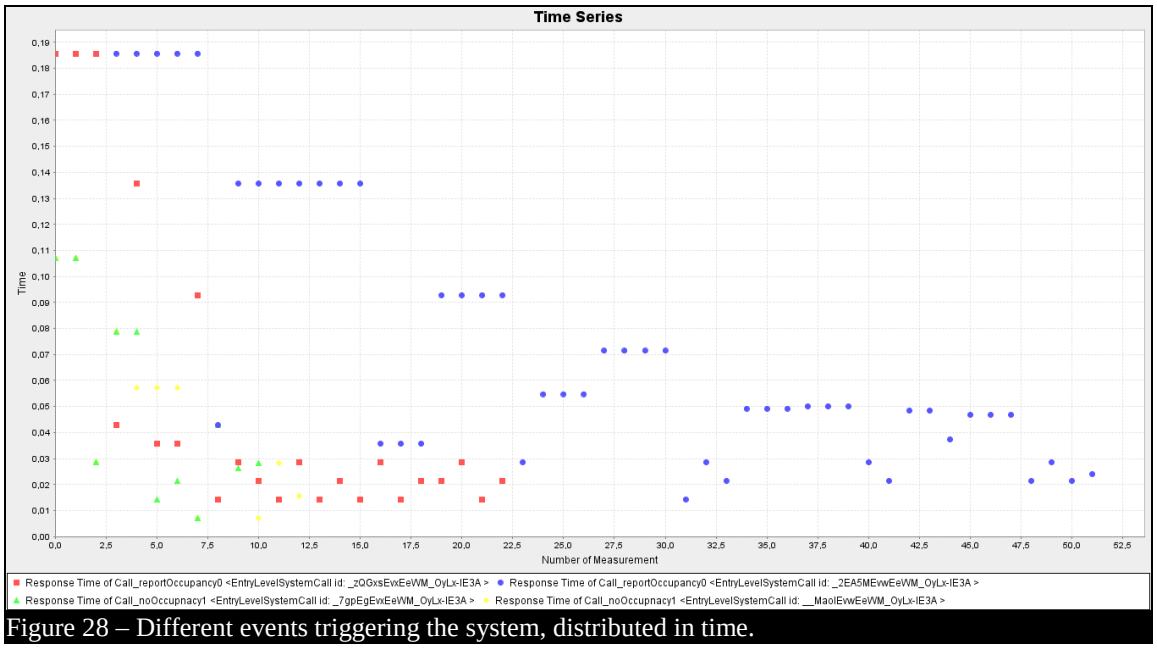

Figure 28 – Different events triggering the system, distributed in time.

9.3.2. Utilization of resource container

Results show also the utilization of resources. For our experiment we have declared one server with CPU that can run 700 cycles/s. Let us see now what results we got

for the resources. The events that we had modelled for the usage scenarios are 10 with thinking time of 1 second for sensor 1, and 5 with thinking time of 1 second for sensor 2. The following pie chart shows the distribution of time among the CPU of our server. Each percentage represents the time that the CPU was busy processing a number of jobs to the whole time the simulation ran. For example, 17.9% of the whole execution time the CPU was busy with processing two jobs (events) that were inside our simulated system.

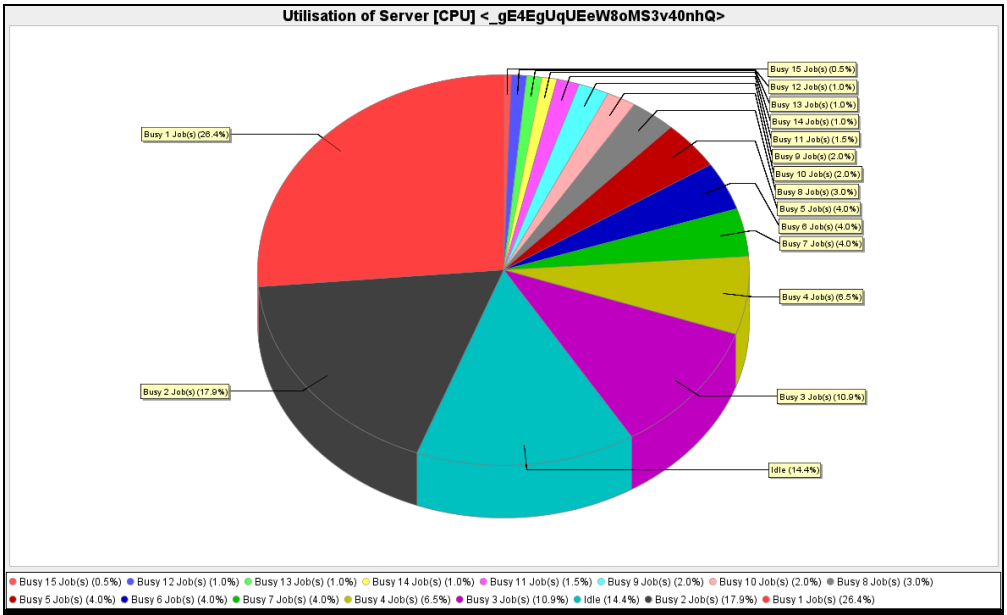

Figure 29 – Utilization of the server (CPU).

9.4 Conclusion

In order to validate and verify our system, we make a small usage example and test it. In this chapter we show all the specification of that system, but also the results we got. The system is rather small for a light system, however it shows that by giving inputs in a specified form, Palladio tool is able to provide results towards the performance of such a system.

We modelled two sensors that triggered by events and give the whole system reason to run. In a real room there will be more sensors and more actuators. Due to time limitation, the experiment minimize some parts.

It is a good input though that having different components as input in the Palladio tool, provide results for each one, even if components are duplicates. Another input from using that tool is that you can have different specification for components that do the same thing. That helps the architect in order to see if for example an expensive sensor in CPU operation will it be a better solution from sensor not that expensive. All are relevant. It is the limits that someone puts to balance performance and quality on a working system.

■

10. Conclusions

Abstract – In this chapter the results of this project are shown. Conclusions of the project are presented, an overview of what has been achieved. Following, is the lessons learned and limitation section that shows different challenges that come across, and finally future work is discussed.

10.1 Conclusions

This project proposes a tool for evaluating architecture candidates, based on reliability criteria. When it comes to reliability, first of all you need to define what reliability for a system is. The evaluation of an architecture is based on key points:

- **Specify use case scenario:** given an architecture candidate, you have to take a use case in that architecture. That helps breaking the architecture into smaller parts, making it easier to evaluate it.
- **Specify components on that scenario:** after specifying the use case, you can specify the components that take place in that use case. That actually is a key point in order to evaluate the architecture. That is because the requirements and the specifications of each component define the evaluation of the architecture composition.
- **Specify reliability:** the definition of the reliability for the system should be reflected on the components. The way the components constructed should eventually when the simulation for the evaluation starts to provide specification for the reliability.

For this project, research made and methodologies for evaluating architecture found. Palladio tool selected to proceed with the experiments, because it was an implemented tool that follows the methodologies. It gives the opportunity to make several scenarios and test them, in a graphical environment.

The main challenge for this project was to construct an architecture from Philips Lighting inside the Palladio tool, in order to provide simulation results. Adding to that the reliability specification should be defined. What was needed to be proved is that with this tool and by following the methodologies for evaluating architecture can actually lead to results that will help architects in their decision. This was achieved in some level as the results in the previous chapter describe.

10.2 Lessons learned and limitations

10.2.1. Lessons learned

Lessons learned from this project, can be categorized, in two categories one contains lessons from working in a big company and second contains lessons more technical in order to provide results.

Gathering requirements, requesting for meetings, and convince people to be part of a project can be really tough. When different people are involved in the project you have to collect requirements from all of them. Basically you have to see the expectations they have, and eventually discuss if you can achieve to meet these expectations.

Requesting for meetings and convincing people to participate in the project is hard. People in a large company tend to be really busy, especially when the people you need are architects. They need good explanation of why what are you making is going to be a helpful tool for them and that their input helps providing a better result.

Architectural documentation is very important, especially when your project has to do with architecture evaluation. I experienced that in StarSense project that at first was the intension to follow. Because there was no documentation it was difficult for an outsider to get familiar with the architecture. Also here the method and tool can help, as it forces the architects to provide information on the architectural composition and usage model. Later on others can use the model to see which compositions were evaluated.

On the more technical aspects, the lessons learned mainly have to do with the architecture specification, and that can be a great input also for architects. When you want to evaluate architecture, you have to be concrete and specific. By following the process described in this project you manage to do that. System's architecture can be huge. You have to break it into smaller pieces, by providing structures for the scenarios that are provided from that architecture. Architecture evaluation is not only the composition you apply to a system, it is a whole process you need to follow.

10.2.2. Limitations

Limitations when using the Palladio tool, and furthermore when using a methodology to evaluate an architecture candidate, has to do with minimum required complexity of the system you want to evaluate. Why is that? The more complex is the system the more dependency parameters you have, which is making simulation run really slow. You have to find the correct balance for that.

Another limitation, also resulting from the above is that the more complex your system, may contain multiple instances of a component. Basically, that means that you have to define everything hard coding in your model.

Time limitations can be seen in Table 12. The values that are shown are for the example used in chapter 9 that was used to validate the tool. It is a rather easy example from the perspective of complexity and modelling. Take into account that when implementing the system in the tool, all information and specification were known.

Table 12 – Time Limitation

Features	Time needed for implementing
Creating components repository	30 minutes
Creating Service specification	30 minutes
Creating System model	30 minutes
Creating resource environment	10 minutes
Creating usage model	10 minutes.

Table 12 shows the time needed for constructing different parts of an architecture and run a simple experiment. If you need to run another simulation, basically you need to change different parts, e.g. create more usage models or different composite structures or different services of each component, but that is something that cannot be defined. If multiple instances need to take part in the composite structure than a simple way to check the time needed is increasing the time spent on the repository modelling and the system modelling with the number of the instances. Moreover, to it's the complexity of the models that change also, connections lines are increasing since there is more connectivity between components, more 'clicks' are needed.

To show the above in a simple example, a composite structure with 25 sensors, one controller, and 50 actuators, needs a lot more clicks in the construction and connecting line between the components then one with only one instance of each component.

10.3 Future work

The need of tools to automate the process of creating the model that is are going to be evaluated. A small example for that is described in the previous sector. Plus, from the Limitation table above we understand that the more complex the model the more time it needs the model to be constructed.■

11. Project Management

Abstract – In this chapter an overview of the project management techniques used for this project is presented.

11.1 Introduction

The Project management strategy used mainly for this project was mainly based on Scrum. This approach is an agile methodology, consists of having close communication with the client.

11.1.1. Scrum approach

Scrum has roles and events, to handle the project management. More explanation of these in Table 9 and 8.

Table 13 – Scrum Roles

Roles	Description
Product Owner	Accountable to represent the stakeholders.
Scrum Master	Accountable for removing impediments to the ability of the team to deliver the product goals and deliverables.
Development Team	Accountable for completing the work.

Table 14 – Scrum Events

Roles	Description
Daily Scrums	The daily scrum meetings is basically stand up meetings to discuss in short time(5-15 minutes) what took place the previous day and what will be the main actions for the current day. For this project there were weekly meetings between Henk and Konstantinos and monthly progress meetings, where Tanir also participated.
Planning Sessions	Meetings where the key points of what have been done to the project is discussed but also evaluated and there comes the planning for what should be the next steps. The planning sessions took place at the beginning of each sprint. Two events take place regards of Planning sessions. The one event is occurring concurrently every week, between Henk and Konstantinos, and it is about what happened, if everything is on schedule and different problems that may arise. Furthermore, there will be also discussion for the goals of the coming week, what are the main

	targets to be full field during the next spring iteration. The second event of planning session is the work that needs to be done. By that we mean, the action points of a previous planning session should have come to the wanted state. Any problems arising should be acknowledged. This actions are done by Konstantinos, and have to do with the sprints results.
Sprint Structure	Each sprint has a duration of one month, and it categorized in four weeks (4 scrums). There is a Week 0, implies that before each sprint probably planning should be done. On every weekly meeting there will be discussed the progress that have been done, deliverable, if any, should be shown, and generally every little progress will be in the agenda. Henk (project owner) will have a clear view of what is going on with the project and will be able to determine the progress of the project and if what have been done connects to the user stories discussed in the planning session.

11.1.2. Other events

In general there are another two events that will take place towards my project.

- Tu/e Progress Meeting: occurs every two weeks between the Tu/E supervisor Tanir and Konstantinos. The discussion is mainly about the progress of the project, and more specific about the quality of the architecture and the documentation that should be delivered along with the product of the project.
- Progress Steering Group Meeting: occurs every last Friday of each month. In that meeting, every stakeholder towards this project can join, and discuss the progress of the project and give ideas and suggestions on how to continue.

11.2 Work-Breakdown structure (WBS)

The initial WBS for the project was done relatively early in the project, in order to be a guideline to follow. It was mainly an estimate, however, it gives insights on the time spent on the specific topics. Since it was an estimation, and knowledge of the domain and the project itself was limited, changes were expected to happen to this plan in later stages. The following Figure 30 shows the initial project planning that was made at an early stage of the project.

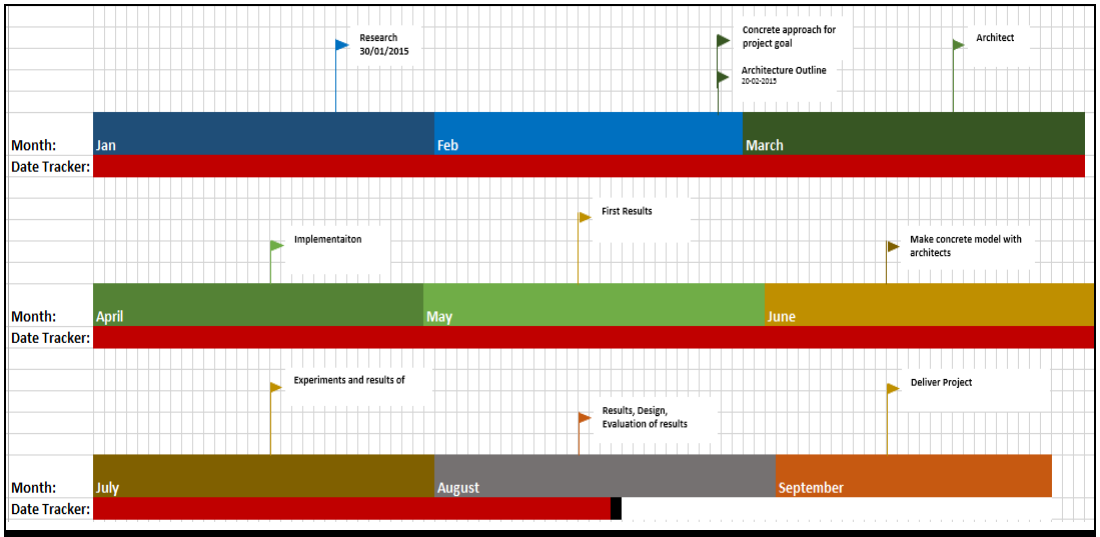

Figure 30 – Project Plan

11.3 Project planning and scheduling

Scrum approach was followed mainly for this project, as section 11.1 describes. The project planning was divided in smaller sprints, in order to break the tasks in smaller and deal with them, towards finding the solution for the project. Sprints have the duration of one month. Figure 31, shows an example of a monthly sprint structure. Clearly, 4 weeks of different tasks are showing.

Sprint Layout January					
	Week 0	Week 1	Week 2	Week 3	Week 4
Activities	Planification	Requirement Capture	Research	Research	Architecture model of StarSense project
		Plan Creation	Validate info from interviews	Finalize Project plan	Make use case scenarios
		Interviews with arch.		Model	Test Scenarios
				Test model	Architecture model of FlexC
					Arcitecture model of LM-IP
					Model based on FlexC/ LM-IP
Deliverables		Feature Plan	Project Plan	Project plan	Project Scope
			Research results	Model	Model
				Test model	Test model

Figure 31 – Example monthly sprint

As time went by, the tasks on each sprint became more concrete. First there was the need to research about the architecture evaluation and familiarize with the domain. Then choose a concrete approach to follow. Table 15 shows the major tasks per sprint iteration.

Table 15 – Tasks per Sprint

Sprint	Tasks
January	Familiarize with Philips Lighting domain. Project Plan Outline. Analysis of the problem. Research. Document chapter 1.
February	Evaluate results from research.

	Gather requirements from architects. Make concrete approach for the project goal. Document chapter 2.
March	Architecture outline. Design decisions. Decision of tool to implement solution. Document chapter 3.
April	Implementation. 2 weeks of vacation. Document chapter 4.
May	Experiment with the tool. First results.
June	Architects involve to make a more specific model for testing. Document chapter 5.
July	Experimenting with the input from the architects, with different projects. Gather simulation results. Document chapter 6 and 7.
August	Evaluation of results. Documentation.
September	Delivery of project. Presentation for the defense.

What can actually be seen from Table 15, is the process that guide to a result for the project. Starting with research, continue with evaluating that results from the research. Experimenting and provide results to architects. Make them want to get more information about the tool and the approach. Make them sit and design and experiment with you and later provide to them the results.

Documentation, was also very crucial for the project. Since it is a 9-month project, everything needed to be documented at a proper time. In every sprint there were slots for writing what has been already implemented. However, by the end of the project documentation was given much more time.

11.4 Conclusions

The creation of a plan posed significant challenges in the beginning of the project. The reasons were the initial ambiguity of the project goals as well as the author's limited domain knowledge. This made accurate estimations very hard. The initial plan was made in order to have a base for the project process and additional refinement had to be done throughout the duration of the project. To conclude, important lessons were learned about the planning process and about the improvement of time and effort estimations while familiarity increases with the domain.■

12. Project Retrospective

Abstract – In this chapter reflection towards the 9-month duration of the projects is made. Looking back to these period can actually see what proved to be good practices; however, what could have done better. Moreover, the design competencies are revisited and their role on the outcome of the project is discussed.

12.1 Good practices

Knowledge and skills that were acquired during the OOTI program, were applicable during this project. From planning techniques, time management up to designing skills. The 9-month assignment at Philips lighting, was an experience that helped you use the knowledge obtained in an industry project.

12.1.1. Make a plan to follow

Making a plan to follow for the very beginning of the project was a very good approach. For sure changes made as long as there was progress with the project; however, creating and follow a plan, makes you work easier, since there is a goal to reach at the end of each week, each month, etc. For me it worked really good, since the beginning was really tough since I had to learn a lot in the domain of Philips lighting, and by making a plan and following, working life became easier.

12.1.2. Contact company supervisors.

Coming into the environment of Philips Lighting, there are a lot that need be learned and the fastest you can do it is better for the process of the project. For me the case was that my supervisor, was always easy to reach to discuss such issues, and also pointing people that were also ones that I should contact to get information.

12.1.3. Creating minutes

Towards the 9 months there were lots of meetings that involved different people. Meetings that important decisions were made, meeting were requirements were discussed. For that purpose I found very crucial for the project and also for me, to create minutes of every meeting that took place regarding my project. It helped me resolve issues and conflicts, and also help people that were involved to keep track of what is happening to the project.

12.2 Improvement points

Now, that the project is finished, by looking back to how things happened and with the experience gained, reflection can be made of things that could have done differently. This section provides that reflection.

12.2.1. Communication with stakeholders

In a project that a lot of people are involved there is the need for good and clear communication. Sometimes I achieved that while others I have not. I got that feeling also from the stakeholders, who sometimes were telling me we do not understand you, you have to be more concrete on what you are saying. What I would do better, plan more meetings, state more clearly what I am doing, which requirements are going to be implemented and which not.

12.2.2. Report issues

Adding to the previous aspect, I would continue with reporting potential issues. Towards the process of finding solution for the project, there were potential issues that I was striving to solve alone, and not report them to anyone of the stakeholders. Sometimes I was succeeding and everything was good, other times I did not succeed. The latter one was causing delays, and stakeholders, due to the fact they haven't been informed about the potential issues, were unhappy about it. I would definitely change that. I would informed better now about any issue, not only because they may cause delays, but also to get help with them.2■

12.3 Design opportunities revisited

In Section 6.3 the design competencies that were deemed relevant to this project at the beginning of the project were described. At the end of the project, these competencies are revisited to determine their fulfillment.

- *Flexibility:* The flexibility aspect, was proven, when implementing an architecture candidate and simulation run, and afterwards, changes on different layers of the candidate needed, in order to re-run simulation. That changes were adapted without any serious problems caused to the parts that were not changed.
- *Reusability*: The reusability aspect was proven, when implementing different architecture scenarios to evaluate candidates. Being able to reuse repositories, already existed, made the time constraints to reduce a lot when introducing a new candidate inside the tool.
- *Complexity*: Complexity was one of the issues that needed to be tackled. This made it an important competency to keep in mind and use for the new design. Indeed, the use of Palladio assisted significantly towards the direction of reducing complexity, toughs making it simple and easy to use domain specific language to model the different aspects of an architecture candidate.

Glossary

Sensor	A sensor is an object whose purpose is to detect motion events in a lighting system
Controller	Gets input from a sensor, that either motion is detected or not. According to that sends message to actuator.
Actuator	Represent the output of the lighting system.
LM-IP	A lighting system implemented by Philips Lighting. Its architecture used to experiment towards this project.
SA	Software Architect
SAE	Software Architect evaluation
SEFF	Service Effect Specification, related to components behavior.
Scrum	An iterative and incremental agile software development framework for managing software projects and product or application development.
OOTI	Onwerpersopleiding Technische Informatica
PSO	Partial swarm optimization
SBRA	Software Based reliability architecture

Bibliography

References

Cockburn, Alistair, Agile Software Development. Cockburn, Alistair and Highsmith, James A. (Eds.), The Agile Software Development Series. Boston: Addison-Wesley, 2002. (0-201-69969-9)

Day, George S., Schoemaker, Paul J.H., and Gunther, Robert E., Wharton on Managing Emerging Technologies. New York: John Wiley and Sons, Inc., 2000. (0-471-36121-6)

Highsmith, James. A., Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York: Dorset House Publishing, 2000. (0-932633-40-4)

Kerth, Norman L., Project Retrospectives: A Handbook for Team Reviews. New York: Dorset House Publishing, 2001. (0-932633-44-7)

Additional Reading

Kelley, Tom, The Art of Innovation. New York: Doubleday, 2001. (0-385-499984)

Adil A. Aziz, Wan M. N. Wan Kadir, A. Yousif, An Architecture-based Approach to Support Alternative Design Decision in Component-Based System: A Case Study from Information System Domain. International Journal of Advanced Science and Technology Vol. 38, January, 2012

V. Firus, S. Becker, Towards Performance Evaluation of Component Based Software Architectures. FESCA 2004 Short Paper

Koziolek, Heiko. Performance evaluation of component-based software systems: A survey. Ladenburg, Germany, 2009.

S. M. Yacoub, B. Cukic, H. H. Ammar, Scenario-Based Reliability Analysis of Component-Based Software.

William W. Everett, Software component reliability analysis.

K. G. Popstojanova, and K. TRIVEDI, Architecture based software reliability.

F. Brosch, Integrated Software Architecture-Based Reliability Prediction for IT Systems.

A. L. GOEL, Software Reliability Models: Assumptions, Limitations, and Applicability

About the Author

Konstantinos Filippidis received his Diploma in Information and Communication Systems Engineering from the Aegean University, Greece in June 2012. During his studies he specialized in Software Technology, Systems Architecture Design and Analysis. His diploma thesis is titled “Design and Implementation of Behavioral-Driven games for modern mobile platforms”. The goal was to create a cognitive Android App employing machine-learning algorithms to interactively adapt to the user gaming expertise, thus offering a more realistic and exciting game experience. In 2013 he joined the two-year PDEng Software Technology program of the Eindhoven University of Technology.

3TU.School for Technological Design, Stan Ackermans Institute offers two-year postgraduate technological designer programmes. This institute is a joint initiative of the three technological universities of the Netherlands: Delft University of Technology, Eindhoven University of Technology and University of Twente. For more information please visit: www.3tu.nl/sai.