Coherent tool support for design-space exploration

3TU. School for Technological Design

STAN ACKERMANS INSTITUTE

Coherent Tool Support for Design-Space Exploration

Fangyi Shi

September 2014

TU Delft
Delft University of Technology

TU/e Technische Universiteit Eindhoven University of Technology

UNIVERSITY OF TWENTE.

Coherent Tool Support for Design-Space Exploration

Fangyi Shi
September **2014**

Coherent Tool Support for Design-Space Exploration

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

Embedded Systems Innovation By TNO Eindhoven University of Technology

Steering Group

Frans Reckers
Bart Theelen
Ad Aerts

Date

September 2014

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
978-90-444-1310-6

Abstract

Embedded Systems Innovation by TNO developed three generic tools to improve industrial applicability. POOSL provides an integrated editing, debugging and validating environment for system modelling, combined with a simulator. TRACE is a tool for visualizing quantitative analysis results. Design Framework (DF) aims for system architecting including architectural views, work flow support and the link of architectural reasoning to concrete modeling activities and artifacts. However, there is no integrated environment to support these three tools to work together.

This project proposes a prototype to demonstrate cooperation between the three generic tools as an integrated environment for design-space exploration. The report describes the process that was applied to the new integrated environment, Exploration Experiment (EE). EE provides a platform to define an experiment by specifying a sequence of a model and executors. From DF, the user can execute a defined experiment and get the execution results automatically.

In addition to the development of EE, this report also includes the process of the development of TRACE extensions. The extensions contain new functionalities of multiple Gantt Chart comparison and design-space visualizations, a standalone application and an executable JAVA Archive file.

Keywords
Design-Space Exploration, Coherent Tool Support, Exploration Experiment, Design Framework, POOSL, TRACE, execution flow, Gantt Chart, Design-Space Visualization

Preferred reference
Fangyi Shi, Coherent Tool Support for Design-Space Exploration. Eindhoven University of Technology, SAI Technical Report, September, 2014. (978-90-444-1310-6)

Partnership

This project was supported by Eindhoven University of Technology and Embedded Systems Innovation By TNO.

Foreword

TNO-ESI strives to improve embedded systems engineering by doing industry-as-laboratory research projects and dissemination of results in several ways. An important form of dissemination is by means of generic model-based design tools for embedded systems. Contributions from industry-as-lab projects have resulted in three generic tools that are actively being professionalized to improve industrial applicability:

1) POOSL	to edit and validate models for discrete-event simulations
2) TRACE	to visualize performance analysis results
3) Design Framework	for system architecting including architectural views, work flow support and the link of architectural reasoning to concrete modeling activities and artifacts

The goal of the assignment "Coherent Tool Support for Design-Space Exploration" was to demonstrate cooperation between the three generic tools as an integrated environment for design-space exploration. Each of the three tools provides an essential cornerstone in a model-based design-space exploration and must remain independent since they can also support embedded system design individually as shown in past and current industry-as-lab projects.

The main challenges for the assignment originated from the need to extend existing tools with new functionalities & interfaces and from realizing a coherent interaction between independent tools. The approach must be robust against the fact that each of the tools is still being developed, which means that limited documentation exists and that adaptations and extensions must be aligned properly.

During the first part of this assignment, Fangyi has realized various improvements of the visualization tool TRACE. She improved the functionality (by new extensions), quality (by removing bugs), flexibility and performance (by finding a much smaller footprint). During the second part of the project, Fangyi has worked in close cooperation with her OOTI colleague and team mate Bayasgalan to design and implement the new Exploration Experiment tool that allows specifying & executing design-space exploration experiments using POOSL simulations and TRACE visualizations, all of which being manageable by the Design Framework.

We enjoyed the fruitful and well-prepared progress meetings and steering group meetings. We look forward to continue our cooperation with Fangyi as our new TNO-ESI colleague after her graduation.

We are grateful to Fangyi for her contributions to TRACE and the Exploration Experiment tool and wish her a lot of success in her future career.

Eindhoven, September 2014

Frans Reckers, Bart Theelen
Embedded System Innovation by TNO

Preface

This report is the graduation report of Fangyi Shi from the Stan Ackermans Institute's Software Technology program of the Eindhoven University of Technology. The report is the result of the project "Coherent Tool Support for Design-Space Exploration". The project has been carried out at Embedded System Innovation by TNO for nine months.

In this report, the design and development progresses for the Exploration Experiment tool and the TRACE tool are described. An detailed explanation of the domain, the problem, the goal and the requirements of this project is provided from Chapter 1 to 4. Readers who are mainly interested in architectural design and technology choices can focus on Chapter 5 and 6. The results, lessons learned and a general reflection of the project can be found in Chapter 7, 8 and 10. The project management progress is described in Chapter 9.

September 13, 2014

Acknowledgements

There are a number of people I would love to express my great gratitude for their support and assistance during the project.

First and foremost, I would love to thank Frans Reckers for giving me the opportunity to carry out this project at TNO-ESI. He was not only a very good project manager, but also a great coach to my personal development.

Also, full of thanks to my company supervisor, Bart Theelen, who devoted a lot of time and effort to guide my direction, challenge my idea, assist my work, review my progress.

Besides, I am very grateful to my university supervisor, Ad Aerts, for his supervision and valuable feedbacks throughout the entire project as well as in the past two years. I became more and more confident thanks to his support and confirmation.

Thanks also go out to my teammate as well as my friend, Bayasgalan Baatar, for her great ideas and efforts. This project would not have been carried out without her hard work.

Furthermore, I would like to thank all the colleagues at TNO-ESI for their assistances and feedbacks. Especially to the Design Framework team, Roelof Hamberg, Peter Vink, Marc Willekens, for their hard work on developing extensions in DF to support this project. Thanks also go out to Jeroen van Schelven for his help during the TRACE extension development.

Special thanks to Maggy de Wert and all the OOTI fellows. I enjoyed every moment that spent together during the two-years OOTI program.

Last but not the least, I would love to express my greatest thanks to my family and friends for their unconditional love, support and encourage during the past two years.

Fangyi Shi
September 14, 2014

Executive Summary

Embedded Systems Innovation by TNO (TNO-ESI) is a leading Dutch research group for high-tech embedded systems design and engineering. It strives to improve embedded systems engineering by doing industry-as-laboratory research projects. Contributions from industry-as-lab projects have resulted in three generic tools that are actively being professionalized to improve industrial applicability:

- POOSL - provides an integrated editing, debugging and validating environment for POOSL modelling, combined with a high-speed simulator. It is used for analysis of system behavior.
- TRACE - is a tool for visualizing quantitative analysis results.
- Design Framework (DF) - aims for system architecting including architectural views, work flow support and the link of architectural reasoning to concrete modeling activities and artifacts.

As today's embedded systems are rapidly becoming more complex, an important challenge in the early stages of the design of embedded systems is the multitude of design possibilities that need to be considered. Design-space exploration is applied for designing these kinds of complex embedded systems. With the help of the three tools, a developer can design complex embedded systems by specifying system requirements with DF, developing and debugging models with POOSL, simulating models with the POOSL simulator and visualizing analysis results with TRACE.

However, there is no integrated environment to support these three tools to work together. Therefore, the main goal of this project is to demonstrate cooperation between the three generic tools as an integrated environment for design-space exploration.

In order to meet the goal, I worked together with my teammate, Bayasgalan Baatar. We designed and developed a new tool, Exploration Experiment (EE), to provide the integrated environment.

In addition to the new EE tool, extensions to the existing three tools are required in order to be integrated with the new functionalities and interfaces. The extensions of the TRACE tool are also part of my responsibility in this project.

The results of this project include:

- The EE tool, which provides a prototype of the integrated environment to demonstrate cooperation between the three generic tools. The EE tool contains a web application and a Java application. The web application is used to define an experiment by specifying a sequence of a model and executors. The Java application is a running server to handle experiment execution requests.
- A new released version of TRACE Eclipse plug-in, which includes new functionalities of multiple Gantt Chart comparison and design-space visualization. In order to integrate with the EE tool, an executable Java Archive file with TRACE functionalities was wrapped. Besides, a standalone application with small footprint was also developed in this project. The standalone application works independently from Eclipse IDE and provides more flexibility for the tool users.

The results from this project prove the concept of coherent tool support for design-space exploration. The integrated environment can be used to the industry-as-lab projects at TNO-ESI in future.

Table of Contents

List of Figures

List of Tables

1.Introduction

This report describes a new proof of concept for the coherence of the set of tools that are developed at TNO-ESI to support design-space exploration (DSE). There are four tools involved: Design Framework, Exploration Experiment, POOSL and TRACE. This chapter introduces the relevant background information, discusses the goal of the project, and provides an outline of the entire report.

1.1 Context

This project is a nine-month project that was provided by Embedded Systems Innovation by TNO (TNO-ESI). TNO-ESI is a leading Dutch research group for high-tech embedded systems design and engineering. It has a close cooperation with high-tech industry, as well as a strong association with fundamental research of academia. TNO-ESI contributes to society and the economy by driving advances in high-tech systems technology, with a strong shared research program, dedicated innovation support, a focused competence development program, and various knowledge and experience sharing activities. See more description in [1].

In order to conduct good research for industry, TNO-ESI develops several tools:

- **POOSL** - provides an integrated editing, debugging and validating environment for system modelling, combined with a high-speed simulator.
- **TRACE** - is a tool for visualizing quantitative analysis results.
- **Design Framework (DF)** - aims for system architecting including architectural views, work flow support and the link of architectural reasoning to concrete modeling activities and artifacts.

More detailed information is in [2].

This project requires the integration of all of the above tools; moreover, a new tool Exploration Experiment (EE) is needed to provide an integrated environment to support the entire progress. The set of these tools are referred to as ESI tools in this report. Except EE, the other ESI tools have already been used by industry independently. In order to support the coherence, all of them need to be extended to allow integration. However, the extensions should not affect the individual usage of the tools. The extensions of POOSL and Design Framework are not the main focus of the project. The TRACE extension and EE are included this project.

Therefore, the entire nine-month project can be divided into two parts:

- Design and development of TRACE extension
- Design and development of EE

The first part is a completely individual work, which is conducted in the first three months. The second part, which lasts six months, is done in cooperation with my OOTI colleague, Bayasgalan Baatar.

1.2 Design-Space Exploration

As today's embedded systems are rapidly becoming more complex, an important challenge in the early stages of the design of embedded systems is the multitude of design possibilities that need to be considered. Design-Space Exploration (DSE) is applied for designing these kinds of complex embedded systems.

The concept of model-based DSE is illustrated in Figure 1. A developer usually starts from a set of concepts and requirements and needs to produce a design that embodies the concepts and satisfies the requirements. This is done in a stepwise process where

in each the design for particular selection of the requirements are explored. To do this, the developer needs to develop and debug models of design alternatives. After several iterations of validating with requirements and modifying models, well-defined models are developed. Then property evaluation can be conducted to get exploration results. After analyzing and interpreting exploration results, the developer may reach final design decisions or may become aware of some issues in concepts and requirements level and then improve the requirements.

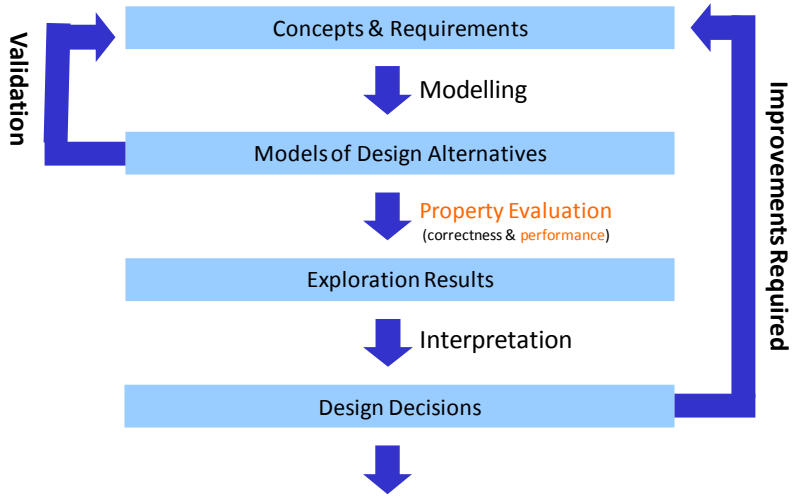


Figure 1 Concept of Model-Based Design-Space Exploration

The ESI tools can be referred to some parts in Figure 1 separately.

- **POOSL** - can be used to develop and debug "Models of Design Alternatives". The POOSL simulator, called Rotalumis, refers to " Property Evaluation " part and the simulation results refer to "Exploration Results" part.
- **TRACE** - is a result visualization tool and is related to the "Interpretation".
- **DF** - keeps the information of "Concepts & Requirements" and "Design Decisions".
- **EE** – provides an integrated environment which controls a flow of “Modelling”, “Property Evaluation” and “Interpretation” by integrating the above three tools to work together to support model-based DSE.

The main goal of the project is to demonstrate cooperation between the three generic tools (POOSL, TRACE, DF) with an integrated environment (EE) for design-space exploration.

1.3 TRACE Extension

Another goal of this project is to extend functionalities in the TRACE tool in order to provide more features for DSE. TRACE was originally used for visualizing a single Gantt Chart.

Two extensions to TRACE are the main focus in this project:

- Multiple Gantt Chart files visualization to support comparison
- Design-Space(DS) Graphs for visualization of statistical data in design-spaces

By adding the two main functionalities, the TRACE tool can provide better features for interpretation in DSE.

1.4 Outline

This report describes the development of the coherent tool support for DSE, as well as the development of TRACE extensions. It is organized as follows:

Chapter 2 provides a domain analysis, where a detailed description of the ESI tools as well as their current development stages are discussed. This chapter also gives more explanation about how the ESI tools should support design-space exploration.

Chapter 3 presents a problem analysis, where the problem to be solved is defined and the goals and design opportunities are identified.

Chapter 4 gives a comprehensive description of the requirements for both EE and TRACE development. Both functional and non-functional requirements are listed.

Chapter 5 describes the EE system architecture and TRACE system architecture separately. Various architectural diagrams are provided in order to explain the systems from different views.

Chapter 6 explains the design and implementation details in both EE and TRACE. The detailed explanation of how to design and implement the systems are addressed. A number of design alternatives are evaluated to make essential technological choices.

Chapter 7 discusses verification and validation methods and results. It focuses on verifying and evaluating the requirements which are defined in Chapter 4. A comprehensive case study is used to validate the EE and TRACE systems.

Chapter 8 concludes the results of the project. Lessons which have been learned during the project are also discussed. It also proposes some new features or improvements for future work.

Chapter 9 offers an insight into the management which has been carried out throughout the project. It discusses project planning and scheduling, work-breakdown structures and risk management.

Finally, Chapter 10 reflects upon the entire process of the project. It also revisits the major design criteria to check whether they have been carried out successfully.

2.Domain Analysis

This chapter gives a detailed introduction of the ESI tools, their current development stages, as well as the relevant technologies involved. Moreover, the concept of how to join these tools to support DSE is also described.

2.1 Current Tools and Relevant Technology

2.1.1. Design Framework

Design Framework(DF) aims at capturing the design rationales in the process of designing embedded or cyber-physical systems. Its principal concepts cover storing the design rationales, which encompasses design decisions and analysis results, by linking design goals to concrete questions and analysis results for a particular scope of the system. The DF does also provide a mechanism for using heterogeneous models for different system parts and linking them by means of essential design parameters and their dependencies. An elaborated conflict detection mechanism at different levels is provided in order to enable the designer to keep the design consistent throughout the process. [14]

A design process of a complex system consists of many activities. These activities can refer to a specific component of the system or parts of this component. These components and their parts are recognized as blocks in DF, and they are organized in a tree structure. Each block can contain a number of parameters. The parameters can be linked as input/output parameters to transformations. The transformation has a model to conduct experiments. If input parameter values are modified, some experiments must be conducted accordingly then the output parameter values can be updated automatically. In the current implementation, it supports Excel, Matlab, and formula models. Besides, the conflict detection mechanism is implemented by the concept of validation. A validation takes parameters as its inputs and confirms whether their values are satisfying the constraints.

Figure 2 shows a simple DF project "demonstrator" with a tree style decomposition. Parameters “M1” and “M2” from two blocks “memory 1” and “memory 2” are connected as inputs to the transformation with a Excel model “test model”, while parameters “latency” and “throughput” from the block “performance” are connected as outputs to the transformation. The green circle indicates the constraint that “M2” is larger than “M1” is validated. By modifying the values of “M1” or “M2”, the values of “latency” and “throughput” will be updated after the execution of the transformation has completed.

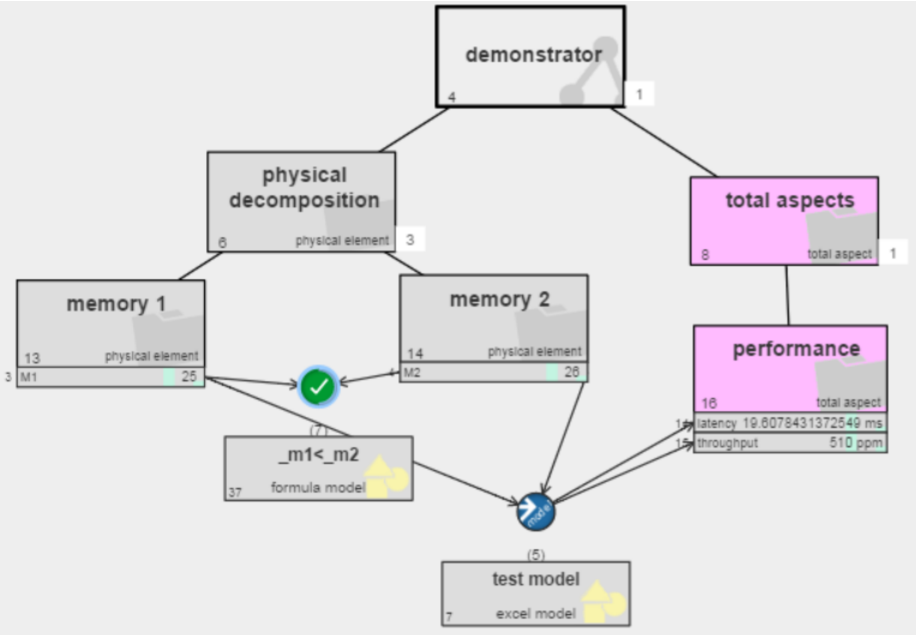


Figure 2 A Simple Use Case in DF

DF has been implemented as a web application, and the main technologies applied are: PHP, Apache server, MySQL database, HTML5, and JavaScript.

See more information about DF in [4].

2.1.2. Exploration Experiment

The Exploration Experiment (EE) tool does not exist yet, and even needs to be designed from scratch. EE is introduced to accomplish some experiments for DF. As mentioned in the previous section, the transformation has a model to conduct some experiments. EE is a tool to define such a model and carry out all of the concrete experiments. Since the concept for EE is new, no relevant technology is specified. Design and implementation are the main part of this project.

2.1.3. POOSL

POOSL is an acronym for Parallel Object-Oriented Specification Language. It is a system-level description language that has been used for modelling complex systems. It has well-defined formal semantics, which is a prerequisite for performance analysis and verification.

As a modelling language, POOSL can specify systems, define cluster and process classes, and illustrate data transmission for any complex embedded system. POOSL models can be simulated by Rotalumis, a simulator that has been developed to conduct a simulation for a well-defined POOSL model. The simulation results can demonstrate how a system reacts for different instantiations, by setting different parameter values inside its corresponding POOSL model.

The POOSL tool contains two parts: a POOSL Editor/Debugger and a POOSL simulator. The technology applied in the POOSL Editor/Debugger is Eclipse plug-in, Xtext and EMF. In this project, only well-defined POOSL models are used, so the POOSL Editor/Debugger is not a concern. However, a POOSL model convertor is needed before starting to run the simulator. The convertor contains two parts, the first part is applied to set parameter values inside the POOSL model, while the second part is to convert the POOSL model to a xml file as an input file for the simulator. The first part has not been developed yet. The second part has already been

wrapped into a ".jar" file, which is derived from the same sources underlying the POOSL Editor/Debugger. The POOSL simulator is an ".exe" file.

The details about POOSL are described in [5] and [6].

2.1.4. TRACE

TRACE is a Gantt Chart visualization tool for any generic activity scheduling. It is capable of presenting large sets of Resource-Claim-Dependency relationships as a function of time. A simple visualization output is shown in Figure 3.

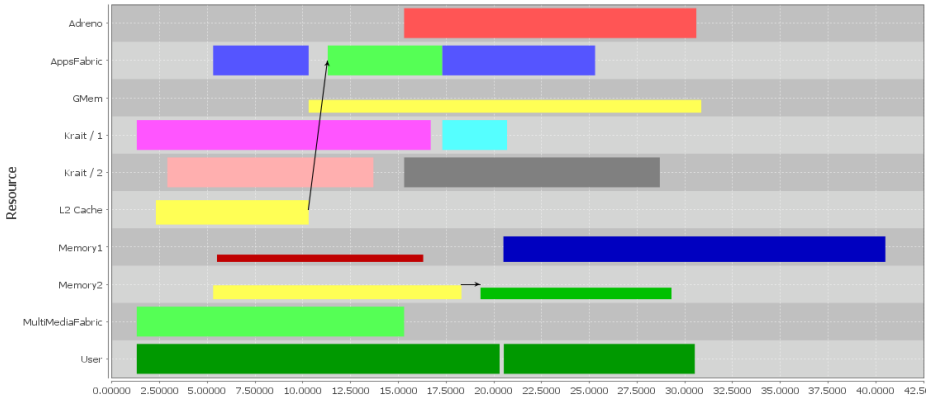


Figure 3 A Simple TRACE Gantt Chart

The input for TRACE is typically the result of analysis tools or simulators in the domain of TNO-ESI applications, such as Rotalumis simulation results. Since the subject of the results can be quite different, the generic TRACE tool is configurable to support identification, selection, and presentation in a way that fits the application area of the subject.

Since the TRACE tool is an Eclipse plug-in, the development language is Java. In order to draw Gantt Charts, it imports the jFreeChart library.

The detailed description and concrete examples are from [7].

One extension to the existing TRACE tool is to include some functionalities to support Design-Space(DS) graphs from another tool, Envisioncy. The Envisioncy tool is a Java application developed by a group of TU/e students with a guidance of TNO-ESI. It is used to display DS graph for statistical data. Currently it supports six DS graphs: Radar Graph, 3D Scatter Plot Graph, 2D Scatter Plot Graph, 3D Heat Graph, 2D Heat Graph, and Parallel Coordinates Graph. It uses OpenGL library to generate these DS graphs.

2.2 Design-Space Exploration

A brief explanation of Design-Space Exploration (DSE) has already been described in Chapter 1. The concept of DSE is applied for designing the complex embedded systems, therefore, the domain of this project refers to any embedded system and embedded system designers are the target users.

The user can design a complex system with the help of the ESI tools. For example, he/she can specify requirements with DF, develop and debug models with POOSL, simulate models with the POOSL simulator and visualize results with TRACE. Currently, all of the above steps can be performed separately. With the EE tool developed in this project, the user can define an experiment, which contains a flow of a POOSL model, the POOSL converter, the POOSL simulator and the TRACE visualizer. From DF, the user can execute a defined experiment and get the visualization result automatically.

The overview of how the ESI tools provide support for DSE is shown in Figure 4.

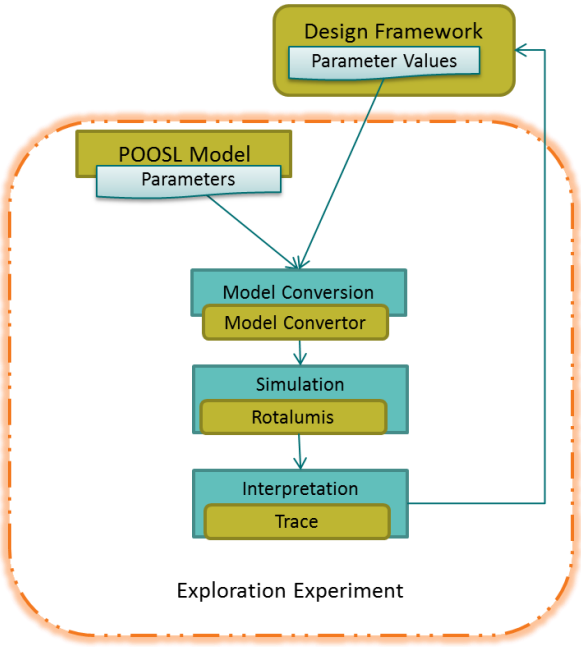


Figure 4 An Overview of Coherent Tool Support for DSE

The figure indicates that:

- Exploration Experiment(EE) contains a POOSL model and provides an environment to conduct an execution flow of POOSL model conversion, POOSL model simulation, and simulation results visualization.
- POOSL is in charge of modelling a system, by specifying applications and platforms with parameterized models.
- DF passes some initial input values to EE and gets the final results from EE. It acts as an external application towards EE and the complex execution details are invisible to DF.
- Model Convertor is used to set parameter values to the POOSL model and convert the model to a format which Rotalumis can handle.
- Rotalumis is responsible for analysis of the specified POOSL model, and produces simulation results.
- TRACE collects the simulation results from Rotalumis and generates relevant Gantt Charts.

3.Problem Analysis

This chapter provides a detailed problem analysis, which involves a concrete problem description, project goal and scope explanation, stakeholder analysis, and an illustration of project challenges and opportunities.

3.1 Problem Description

The purpose of this project is to demonstrate the coherent tool support for DSE. The fact that most ESI tools are still under development increases the complexity level of integration. Well defined interfaces are needed among the tools.

According to the overview of how the ESI tools should integrate to support design-space exploration in Chapter 2, DF is supposed to act as an external application regarding the entire experiment execution. It only deals with the initial input data and the final output data. The interfaces between EE and DF need to be compatible with the current implementation of DF. Before executing the experiment, DF needs to know the definition of the input and output parameters. Therefore, EE has to extract input parameter information from the POOSL model, as well as the output parameter information from the TRACE tool for DF.

After getting the input and output parameter information from EE, DF can set values for input parameters and trigger the execution. EE should collect these values from DF, map these input values to the relevant POOSL models, and then conduct the simulation and visualization processes. Once the entire process has been done, EE should update the final TRACE results in DF.

3.2 Project Goal and Scope

As it is mentioned in Chapter 1, this project can be divided into two parts: TRACE extension development part and EE development part. The first part is one person's three-month work, while the second part is two persons' six-month work. Therefore, regarding time duration, my main effort is on the development of the EE tool.

The main goal of this project is to demonstrate the ESI tools working together to support DSE, by fulfilling the following functional sub-goals:

- Define and develop interfaces between DF and EE, which requires cooperation with the DF team.
- Define and conduct execution flows in EE, which needs cooperation with my teammate.
- Develop TRACE extensions to support more visualization features, which is my individual work for three months.
- Develop POOSL extensions to support getting and setting parameters, which belongs to the scope my teammate.
- Develop DF extensions to support interactions with EE, which is part of the DF team's work.

The concrete requirements are discussed in Chapter 4.

There are also some non-functional sub-goals in this project:

- Research various technological alternatives to make technology choices.
- Apply some useful architecture and design patterns.
- Adapt to the company's environment and improve communication skills.

3.3 Stakeholders

The stakeholders involved in this project play different roles and have their specific concerns. Table 1 lists the various stakeholders' concerns.

Table 1 Stakeholders and Concerns

Name	Role	Concerns
TU/e (OOTI)	Conductor of the OOTI final project	• Complexity of the design • Difficulty of challenges • Quality of the final report
TNO - ESI	Owner of the project	• Progress of the project • Sufficient manuals • Ready to use tools
Users	End users of the products	• Ease of use • Good performance
Software Developers	Developers of the ESI tools	• Ease of integration • Loose coupling among the tools • Maintainability
Project Teammate	Bayasgalan Baatar, the other OOTI trainee working on this project	• Clear scope of individual work

3.3.1. The University

The university (TU/e) focuses on the learning curve of this project and the overall architecture and design. Besides a good architecture for the entire project, the university also aims for well-written academic documentation.

3.3.2. The Company

TNO-ESI, as the owner of the project, focuses on the feasibility of the prototype. Moreover, the available documentation of the ESI tools is to be improved substantially; hence this stakeholder requires sufficient documentation, such as user manuals and development documents.

3.3.3. End Users

Regarding the ways of using the ESI tools, end users can be separated into two groups.

One group uses these tools individually, and they have already had experience with the existing versions. They have formed their habits of using these tools. When new features are introduced, they do not easily accept big changes. The users in this group also can have different demands. For example, the users from one TNO-ESI partner require an Eclipse plug-in for the TRACE tool, because the company's policy of installing new software. However, the users from another TNO-ESI partner do not prefer the Eclipse plug-in, because there are many redundant UI features in Eclipse where they only need to use the TRACE visualization features.

The other group of users is to use the coherent tool support for design-space exploration. They are test users for this project, and their concern is a user-friendly interaction among these tools.

3.3.4. Software Developers

Three tools, DF, POOSL, and TRACE, are still under development. The developers' main concern is to keep these tools loosely coupled and ensure the minimum impact of changes in the individual tools. Therefore, it is a challenge to ensure the fewest integration points.

Moreover, in order to make the tools work together, we need the input from the tool developers. The developers of DF provide us an idea of how DSE can be controlled through their user interface. POOSL developers provide us some functionality to make POOSL extension easier. TRACE developers need to transfer all the software code and explain it in detail. Then, the progress of extending TRACE functionalities can move on smoothly.

3.3.5. Project Teammate

Since this project is a two-person project, we need to divide our tasks clearly and work together with each other.

3.4 Opportunities and Challenges

Design opportunities and implementation challenges for the EE development part differ from those for TRACE development part.

3.4.1. Opportunities and Challenges in EE

There are many opportunities in the EE design and implementation:

- Aim for a generic EE
  According to the problem description, EE can be a part of DF and can also work individually. Loose-coupling between DF and EE is a big concern during the design. In this project, EE only needs to support POOSL models, the POOSL convertor, Rotalumis and TRACE. However, the design of EE needs to be as generic as possible. In further, EE should be easily extensible to support other models and executables.
- Communicate with DF
  EE interacts with DF, which is a web application. The methods to communicate with a web application are plenty, such as web sockets, web services, and TCP/UDP sockets. The evaluation of these technologies is a part of the design.
- Deploy and run all the executables
  Multiple executables should be executed inside EE in parallel. Some of them might take a long time, so a stable environment is preferable to support a long execution.
- Define the execution flow and relevant models
  In order to refer to specific models and organize the execution flow in an adequate way, it is necessary to have a well-defined experiment. How to define and organize the experiments is also a challenge in this project.
- Vary among technologies
  EE needs to work with different tools, which are implemented in different programming languages. Therefore, EE also relies on various technologies at the same time:
    - web application technologies, such as HTML5, PHP, JavaScript
    - database technologies
    - web socket and web service technologies
    - programming languages, such as Java or C++

3.4.2. Opportunities and Challenges in TRACE Extension

The current version of the TRACE tool has a simple structure without any software architecture pattern or design patterns. There is a good opportunity to refactor the code and apply some design patterns and a proper architecture.

Unlike EE design and implementation, the TRACE tool requires a deep knowledge of Eclipse plug-in development and Java programming skills.

Envisioncy is a Java application and displays DS graphs. In order to show the DS graphs, it has its own framework, such as user interfaces and a file loading mechanism. While merging the Envisioncy functionalities into TRACE, only the code used for generating DS graphs is useful. Therefore, another challenge in extending TRACE is to extract the necessary code from Envisioncy.

4.System Requirements

After a concrete analysis for the relevant domain and problem, the detailed system requirements are discussed in this chapter. Since the entire project can be divided into two parts: EE development and TRACE extension development, the requirements are also separated into two parts accordingly.

4.1 Exploration Experiment Requirements

The requirements of EE include functional and non-functional requirements. The functional requirements focus on the specific features to be supported by EE, while the non-functional requirements specify criteria to be used to judge the operation of the system.

4.1.1. EE Functional Requirements

Three high-level features comprise the EE functional requirements:

- Define an experiment by specifying components (executors and models) and relationships between these components.
- Execute a well-defined experiment by executing a execution chain.
- Interact with DF by taking input values and setting output results.

The detailed functional requirements are shown in the following tables. [1]

Table 2 Requirement - Define an Experiment

ID	Name	Description
A1	Create an experiment	Give a name for the new experiment and get a unique ID from server.
A2	Retrieve an executor	Select an executor from the server and display the executor's information on the UI.
A3	Upload a model	Upload a well-defined POOSL model to the server and display the model's information on the UI.
A4	Retrieve a model	Select an uploaded POOSL model from the server and display the model's information on the UI.
A5	Add a connection from a model to an executor	Specify that a model is an input to an executor from the UI and add this relation to the database.
A6	Add a connection from one executor to another executor	Specify that the output of one executor is an input for another executor from the UI and add this relation to the database.
A7	Download an EE model	After defining an experiment, generate an EE model for the experiment and download it from the server.
B1	Retrieve an experiment	Select an available experiment from the server and display the defined executors, models and relations.
B2	Run an experiment	Start an experiment by passing the current experiment ID.
C1	Display progress status or error messages	Collect the messages from the EE Execution Handler and display them from the UI.

[1] For all the Requirement Specification Tables in this chapter: the priority is identified by ID. High priority requirements begin with "A", medium priority requirements begin with "B", and low priority requirements begin with "C".

Table 3 Requirement - Execute an Experiment		
ID	Name	Description
A8	Generate a run ID	Generate a unique ID for each experiment execution.
A9	Set input values	Parse input information and map the input values to the relevant model.
A10	Create output folders	Create a unique output folder for every executable in the runtime.
A11	Set input arguments	Set the specific input arguments for every executable.
A12	Get the next executor	Look up the database and return the next executor information.
A13	Run an executor	Run an executable with the specified inputs and output path.
A14	Extract parameters	Extract input and output information from one experiment definition.
A15	Generate an EE model	After extracting parameters, generate an EE model accordingly.
A16	Update final results	After finishing all the executions, notify the UI to display the final results.
B3	Execute an experiment with default data	Execute an experiment without mapping the values of input parameters to the model.
B4	Check execution time	Display execution time for every executable.
C2	Update intermediate results	After finishing one execution, notify the UI to display the execution results.
C3	Notify error messages	Notify the UI when there is an exception during the executions.

Table 4 Requirement - Interact with DF		
ID	Name	Description
A17	Upload an EE model	Select an EE model and upload it to the DF server.
A18	Parse experiment parameters	Get experiment parameters from an EE model and convert the parameters to the DF format.
A19	Set input values	Enter experiment parameter values through the UI of DF.
A20	Start an experiment	Start an experiment by passing the experiment ID and input parameter values.
A21	Display output results	Collect the output parameter values or images from the EE and display them from the UI of DF.
C4	Display progress status or error notifications	Collect messages from the EE and display them from the UI of DF.

4.1.2. EE Non-functional Requirements

Besides the functional requirements, it is also important to clarify the essential non-functional requirements. The design of the overall architecture is mainly impacted by the non-functional requirements. Since the opportunities and challenges of EE have already been mentioned in Chapter 3, the following four non-functional requirements are included:

- Loose-coupling
- Generality
- Usability
- Extensibility

Loose-coupling

Loose-coupling refers to how to define minimal interfaces between DF and EE. The purpose of loose-coupling comes from two aspects:

- DF: DF has its own concept of transformation. EE is considered as a supporting tool for conducting experiments and completing the transformations in DF. DF does not need to know how EE works, but only input parameters and output results. Extraction of only necessary EE information for DF is a requirement.
- EE: EE performs as a part of DF and conducts experiment for DF. However, EE is also supposed to work independently from DF. The user should be able to conduct an experiment from the EE UI.

Generality

Generality refers to how a generic EE framework can be built to support arbitrary models and executables. The initial idea of this project is to import POOSL models, execute the POOSL convertor, the POOSL simulator, and the TRACE visualization tool in a sequence. However, EE is also supposed to import other types of models, like Excel models, Matlab models, and run their simulators in the future.

Usability

Usability is related to user's experience with EE. In order to define an experiment, the user needs to specify all the detailed information. The need of user interaction brings up the usability requirement. In general, usability can be judged from two aspects:

- Learning curve: if the users are not familiar with the EE tool at all, they should be able to start a simple use case quickly.
- User interaction: the users can accomplish their tasks with minimal effort and no redundant procedures are required.

Extensibility

Extensibility is related to the generality non-functional requirement. Since EE aims for support for different models and executables, an important issue is how much effort a developer needs to make when introducing other models and executables. A well-structured EE architecture and a good development manual can reduce the developer's effort.

4.2 TRACE Requirements

Apart from the EE requirement specifications, there is another independent group of specifications for extending the TRACE tool.

4.2.1. TRACE Functional Requirements

The TRACE functional requirements are mainly composed of four parts:

- Build a TRACE standalone application, which can work properly without the Eclipse framework.
- Display Gantt Chart comparison for multiple input files.
- Merge Envisioncy functionality into the TRACE Eclipse plug-in to display Design-Space (DS) graphs, and support proper navigation from DS graphs to the corredsponding Gantt Charts.
- Build an executable file for executing from the command line and generating images for Gantt Charts or DS graphs. This executable file is used inside the EE tool to support design-space exploration.

The detailed functional requirement specifications are shown in the following tables.

Table 5 Requirement - Build a Standalone Application		
ID	Name	Description
A22	Include the same functionalities as Eclipse plug-in version	All the functionalities that can work with Eclipse plug-in version can be conducted with the standalone version.
A23	Work independently from the Eclipse IDE	The standalone version can work independent of the Eclipse IDE.
B5	Support multiple operating systems	The standalone version should work properly under different operating systems, such as Windows, Linux, Mac OS.

Table 6 Requirement - Display Gantt Chart Comparison		
ID	Name	Description
A24	Open a Gantt Chart comparison editor	Display a Gantt Chart comparison editor by selecting multiple files or folder(s) as inputs.
A25	Merge Gantt Chart files into one graph	Parse multiple files and represent Gantt Chart comparison within one graph.
A26	Sort claims	Sort all the claims by resource names.
A27	Support some major features for a Gantt Chart Comparison	Support the functionalities for comparison graphs: • Zooming and panning • Filtering • Grouping • Coloring
A28	Show properties of a point	Select a point in a Gantt Chart and display its properties.
A29	Support the existing functionalities for a single Gantt Chart	By introducing the comparison features, the existing features of a single Gantt Chart are still available.

Table 7 Requirement - Display DS Graph		
ID	Name	Description
A30	Open a DS graph selection dialog	Display a DS graph selection dialog by a DS graph request with multiple files or folder(s) as inputs.
A31	Parse quantity files	Get the quantity information from the multiple quantity files and fill the data structure.
A32	Open a DS graph	Display a DS graph with an editor.
A33	Show properties of a point/curve	Select a point/curve in a DS graph and display its properties.
B6	Navigate from a DS graph to a Gantt Chart	Interact with a DS graph or the property view and navigate from the current DS graph to a Gantt Chart graph.
B7	Display a navigation selection dialog	Show a dialog to specify which single Gantt Chart or multiple Gantt Chart comparison needs to display.

Table 8 Requirement - Build an Executable File		
ID	Name	Description
A34	Wrap an executable file	Build a JAR executable file from the TRACE plugin.
A35	Execute the executable	Open the "Command Prompt" and run the executable file by specifying the input arguments.
A36	Generate images	Generate the relevant ".png" or ".jpg" images for Gantt Charts or DS graphs.

4.2.2. TRACE Non-functional Requirements

Three non-functional requirements are taken into account while extending the TRACE tool:

- Usability
- Performance
- Extensibility

Usability

Usability refers to how easy the TRACE tool is for a user to accomplish a task. Since this tool requires a lot of user interactions, usability is a main non-functional requirement. Five aspects are focused on:

- Learning curve: it is the same as the EE tool. If the users are not familiar with the tool at all, they should be able to start a simple use case quickly.
- User experience: since the tool has already been used in some companies, the new features need to keep a similar user experience.
- User interaction: the users can accomplish their task with minimal effort and no redundant procedures are required.
- User styles: the tool is supposed to provide different ways to precede the same task. In this way, the users do not have to be restricted to one specific interaction method.
- File organization: the tool needs to take multiple inputs and these input files are related with each other. A flexible way to organize the files is required.

Performance

Performance refers to how much time the TRACE tool needs to respond to a desired task. As it is an Eclipse plug-in application, the Eclipse IDE consumes most of the time. Nevertheless, the response time still should be optimized, especially towards the following two aspects:

- User setting initialization: if multiple files share the same configuration, the tool needs a minimum initialization time for the same configuration setting.
- File Loading time: since the size of input files can be very large, a way to load those files in an efficient way is required.

Extensibility

Extensibility refers to how easy the TRACE tool is for a developer to add new features. Because the tool is continuously under development by different developers, a well-structured architecture is demanded for achieving extensibility, regarding to two aspects:

- Existing features: if developers need to modify existing features, how easy is it for them to understand the current structure.
- New features: since it is an Eclipse plug-in application, each new plug-in extension can be added separately. Even so, a clear way to merge new functionalities into its logical place in the existing architecture is needed.

5.System Architecture

After investigating all of the key requirements in the previous chapter, this chapter discusses how the system architecture fulfills the various requirements (both functional and non-functional). In order to do so, the 4+1 architecture view model is applied to illustrate Exploration Experiment(EE) and Trace architectures. The 4+1 architecture view model consists of the logical view, development view, process view, physical view, and use case scenario.

5.1 Introduction

The system architectures that refer to EE development and TRACE development are completely independent. Therefore, the rest of the chapter is divided into two parts to illustrate the EE architecture and TRACE architecture. Coincidently, both architectures apply the Model-View-Controller (MVC) architecture pattern. It is a good opportunity to practice MVC with different use cases and development technologies.

Different UML diagrams can be applied to illustrate different views of the 4+1 architecture view model. The corresponding UML diagrams and views are as follows:

- Logical view: class diagram
- Development view: component diagram
- Process view: activity diagram
- Physical view: deployment diagram
- Use case scenario: use case diagram

5.2 Exploration Experiment Architecture

In this section, the EE use case scenarios, the EE development view, EE process view, EE physical view and EE logical view are applied to illustrate the EE overall architecture.

5.2.1. EE Use Case Scenarios

The EE application provides the user a platform to define an experiment. It also generates EE models for DF. An EE model contains the information of input and output parameters for a specific experiment. Based on this EE model, DF can communicate with the EE tool and conduct the relevant experiment.

According to the EE functional requirements, which are discussed in Chapter 4, there are three user goals:

- Define an experiment
- Add an EE model to a DF transformation
- Run an experiment

The use case diagram is illustrated in Figure 5, and followed by concrete use case scenario tables.

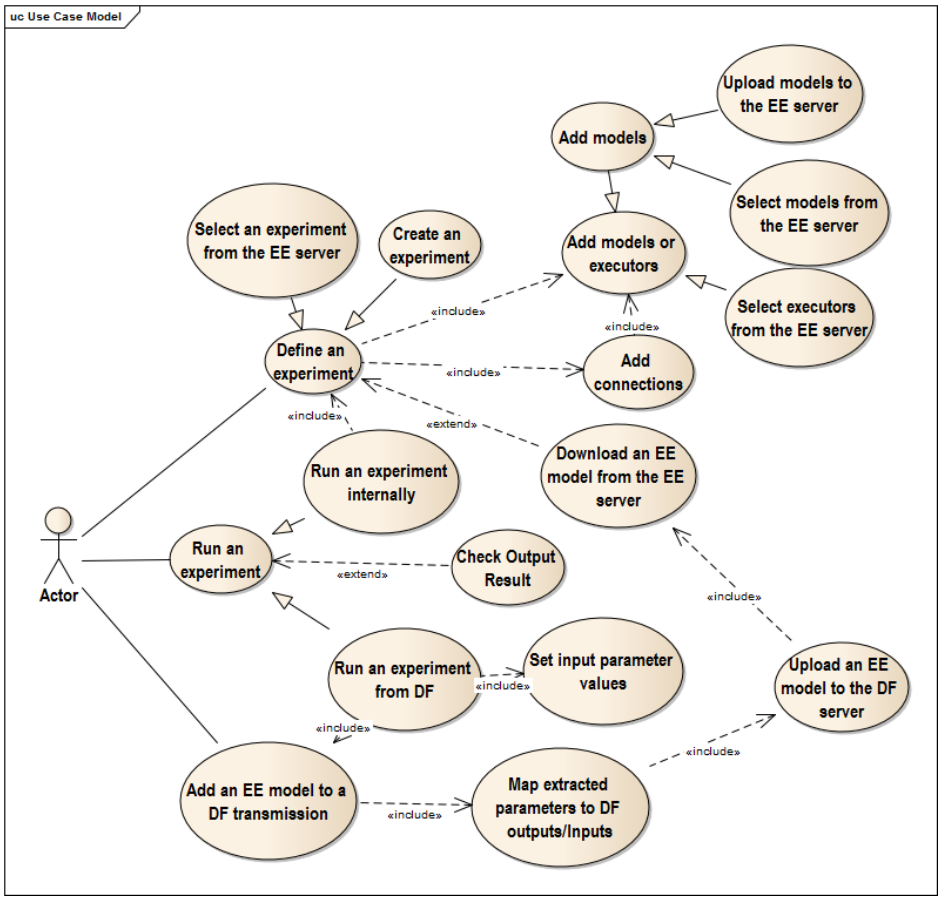


Figure 5 EE Use Case Diagram

Use Case 1: Define an Experiment

Primary Actor: EE Tool User
Context of use: the user wants to define an experiment
Scope: EE Web Application
Precondition: Apache server is on and direct to the EE web page
Success Guarantees: the user can define an experiment and the system can display the defined experiment on the UI and store it on the EE server.
Main Success Scenario
1: EE Tool User: Create a new experiment by giving a name.
2: EE System: Display a blank definition field accordingly.
3: EE Tool User: Select an executor from the EE server.
4: EE System: Display the selected executor accordingly.
5: EE Tool User: Select an executor and retrieve a model from the EE server as an input to the executor.
6: EE System: Display the added model and connect the model to the selected executor.
7: EE Tool User: Select an executor and add another executor as its next executor.
8: EE System: Display the added executor and connect it to the selected one.
9: The order of Step5 and Step 7 can be reversed.
10: Step 5 and Step 7are repeated until all models and executors are added.
Alternate flows
1.a: EE Tool User: Select a defined experiment from the EE server.
EE System: Display the pre-defined executors, models and their connections on the definition field.
5.a : EE Tool User: Upload a model to the EE server.
EE System: Request the model information from the user.
EE Tool User: Enter the model information.

EE System: Save the uploaded model on the server, display the added model and connect it to the selected executor.
Extensions
10.a: EE Tool User: Download an EE model from the EE server.
EE System: Generate the required EE model and display it on the EE UI.
EE Tool User: Save the generated EE model to a local place.

Use Case 2: Add an EE model to a DF transmission

Primary Actor: DF Tool User
Context of use: the user wants to apply a well-defined experiment to DF
Scope: DF web application
Precondition: Open a DF project
Success Guarantees: the user can apply an experiment to DF and the DF system can extract the relevant information of the experiment.
Main Success Scenario
1: DF Tool User: Upload an EE model to a DF transformation.
2: DF System: Extract the parameter information from the EE model and display it.
3: DF Tool User: Map the extracted parameter information to DF inputs and outputs.
4: DF System: Display the mapping results.

Use Case 3: Run an Experiment

Use Case 3.1 Run an Experiment internally

Primary Actor: EE Tool User
Context of use: the user wants to run an experiment from the EE web application
Scope: EE web application and a running EE server
Precondition: Define an experiment
Success Guarantees: the user can run a defined experiment internally and the EE system can conduct the experiment successfully.
Main Success Scenario
1: EE Tool User: Define an experiment, see Use Case 1.
2: EE System: Display the defined experiment on the definition field.
3: EE Tool User: Run the defined experiment with default input parameter values
4: EE System: Execute the desired experiment on the EE server and send the execution results back.
5: EE System: Display the results from the EE UI.
Extensions
5.a: EE Tool User: Check the results through the EE UI.

Use Case 3.2 Run an Experiment from DF

Primary Actor: DF Tool User
Context of use: the user wants to run an experiment from DF
Scope: DF web application and a running EE server
Precondition: A DF project is opened and an EE model has been applied to DF
Success Guarantees: the user can run an applied experiment and the DF system can conduct the experiment successfully.
Main Success Scenario
1: DF Tool User: Apply an experiment to DF, see Use Case 2.
2: DF System: Display the mapping of input and output parameters.
3: DF Tool User: Set input parameter values.
4: DF System: Save the values on the DF server.
5: EE System: Execute the desired experiment on the EE server and send the execution results back.
6: DF System: Display the results from the DF UI.
Extensions
6.a: DF Tool User: Check the results through the DF UI.

5.2.2. EE Development View

From software designer's perspective, the development view uses component diagrams. The component diagram illustrates how the tool components are coupled together. Every tool contributes differently to the system.

POOSL Editor/Debug is used for specifying a model before using it in an experiment. The POOSL convertor, Rotalumis and TRACE are the executors that can be run from the command line. These executors are executable files which take input files and produce output files. How to assemble these executors is inside the integrated environment provided by EE.

Core Components

Since there are three EE use case scenarios, three core components are introduced accordingly. They are EE Definition Handler, EE Execution Handler and Design Framework (DF). The overview of these three core components are in Figure 6.

EE Definition Handler

It is an application to define concrete experiments. It deals with the user interfaces and database management.

EE Execution Handler

It focuses on the logical rules of handling all the executions and also manages the database and file system.

Design Framework

The implementation of the DF component is part of DF team's tasks. However, interfaces between DF and EE Execution Handler need to be well defined by us.

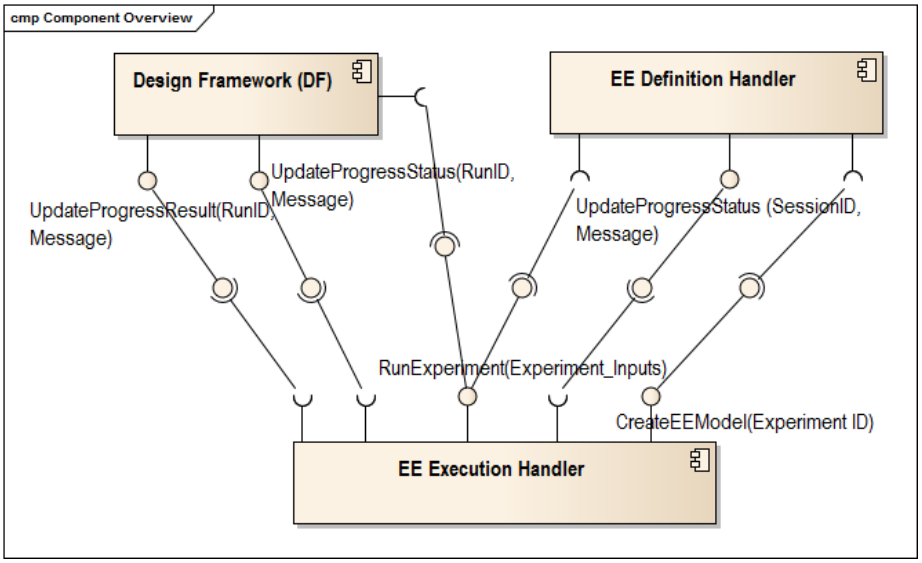


Figure 6 Component Diagram of Core Components

Interfaces among components

EE Execution Handler communicates with DF and EE Definition Handler separately. Figure 6 also defines the interfaces which support these communications.

In order to make as few modifications as possible in the current DF implementation, two types of interfaces are involved between DF and EE Execution Handler.

- Runtime functional call interfaces

These interfaces refer to *UpdateProgressResult*, *UpdateProgressStatus* and *RunExperiment* in Figure 6. *RunExperiment* is provided by EE Execution Handler and invoked by DF, when DF needs to start one experiment. *UpdateProgressResult* and *UpdateProgressStatus* are provided by DF and called by EE Execution Handler. When an experiment has completed, EE can invoke *UpdateProgressResult* to send the output data back to DF. There are several steps to complete an entire execution chain, when one step has completed or an error has occurred during the execution, EE can invoke *UpdateProgressStatus* to update the current progress status.

- File interface
  File interface refers to a special EE model that is defined in EE Definition Handler, generated by EE Execution Handle, and used by DF. In an EE model, it includes the information of experiment ID, input parameters and output parameters. By uploading an EE model to DF, DF can map input/output parameters, set input parameter values, trigger one execution of a specified experiment, and wait for updating output parameters.

The interfaces between EE Definition Handler and EE Execution Handler are three functional call interfaces.

- *RunExperiment* is provided by EE Execution Handler and invoked by EE Definition Handler. It supports the function of executing an experiment with default parameter values of a model.
- *UpdateProgressStatus* is provided by EE Definition Handler and invoked by EE Execution Handler. It updates the progress status for an experiment execution.
- *CreateEEModel* is provided by EE Execution Handler and is called by EE Definition Handler, when the user needs to create an EE model for a defined experiment.

EE Internal Components

Since the development of EE is the main focus of this project, the EE Internal components are discussed in detail in this section.

Defining and Executing an experiment are conducted by EE Definition Handler and EE Execution Handler separately. Defining an experiment requires a graphical user interface(GUI) to allow users to interact. By interpreting users' inputs, EE Definition Handler can update the user interface on the client or manipulate data on the server. Executing an experiment does not contain a user interface, but and EE Execution Handler takes inputs from DF or EE Definition Handler, conducts the execution flow, produces various outputs, modifies the data in the database, updates progress status and updates results.

Although EE Definition Handler and EE Execution Handler are separate components, the overall architecture is applied by regarding the two components as one EE component. As usability and extensibility are two important non-functional requirements for EE, a framework which can well organize users' inputs, control data model, and present views is preferable. Therefore, the Model – View – Controller (MVC) architecture pattern is selected.

The MVC pattern separates the entire architecture into three main parts:

- **Model** contains application data, business rules, logic and functions.
- **View** can be any output representation of information, such as a chart or a diagram.
- **Controller** interprets the inputs from the user, such as mouse and keyboard actions, manipulates the model, and updates the view.

A detailed component diagram of EE internal components is shown in Figure 7.

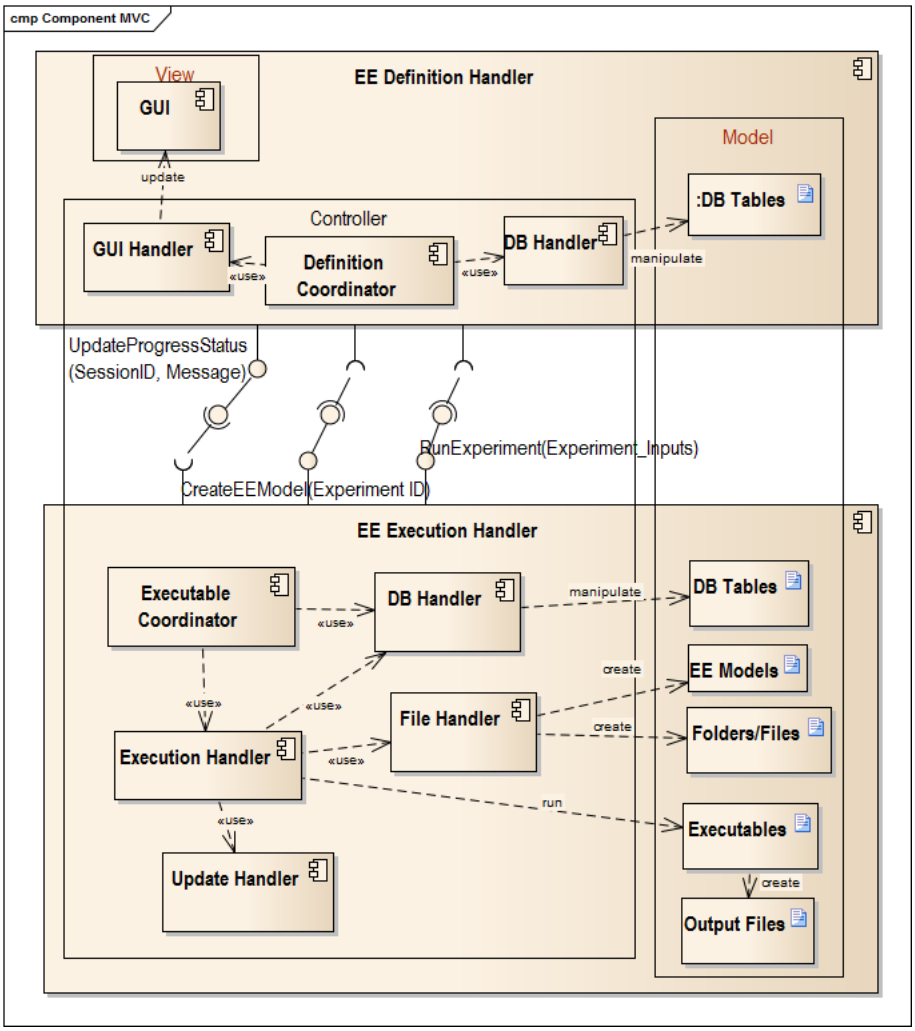


Figure 7 Component Diagram for EE

The concrete role of each component according to MVC is explained below:

Model

The model part mainly includes files and database tables, which are mainly manipulated by different handlers.

- DB Tables: database tables, which contain all the EE definition information, executable paths and some runtime information. EE Definition Handler and EE Execution Handler share the same database tables.
- Executables: they are executable files, which can be run through the command line.
- EE models: the files contain all the information of input and output parameters.
- Files/Folders: the files or folders are created in the runtime.
- Output files: they are the generated by the executables.

View

The user only interacts with EE Definition Handler, which includes a GUI view part. The GUI can display EE definition information by drawing a flow of a model and executors on the canvas. It can also show a runtime status when conducting an experiment.

Controller

The controller can be separated into two parts, one belongs to EE Definition Handler and the other belongs to EE Execution Handler.

The controller part inside EE Definition Handler:

- GUI Handler: it takes the user's actions and updates the GUI.
- DB Handler: it connects to the unique EE database from EE Definition Handler side.
- Definition Coordinator: it is a coordinator, which is in charge of internal communication with the EE Execution Handler. It uses the GUI Handler and DB Handler to define an experiment.

The controller part inside EE Execution Handler:

- Execution Handler: it controls the execution chain and executes different executables.
- DB Handler: it connects to the unique EE database from EE Execution Handler side.
- File Handler: it deals with file or folder modification in the runtime.
- Update Handler: it is in charge of updating results back to the EE Definition Handler or DF.
- Executable Coordinator: it is a coordinator of this controller part. It communicates with EE Definition Handler or DF, and guarantees a request can be performed properly.

5.2.3. EE Process View

After a general description of the three core components, the overall activity diagram is shown in Figure 8. It illustrates the action flows in the system as well as the interactions among the three components.

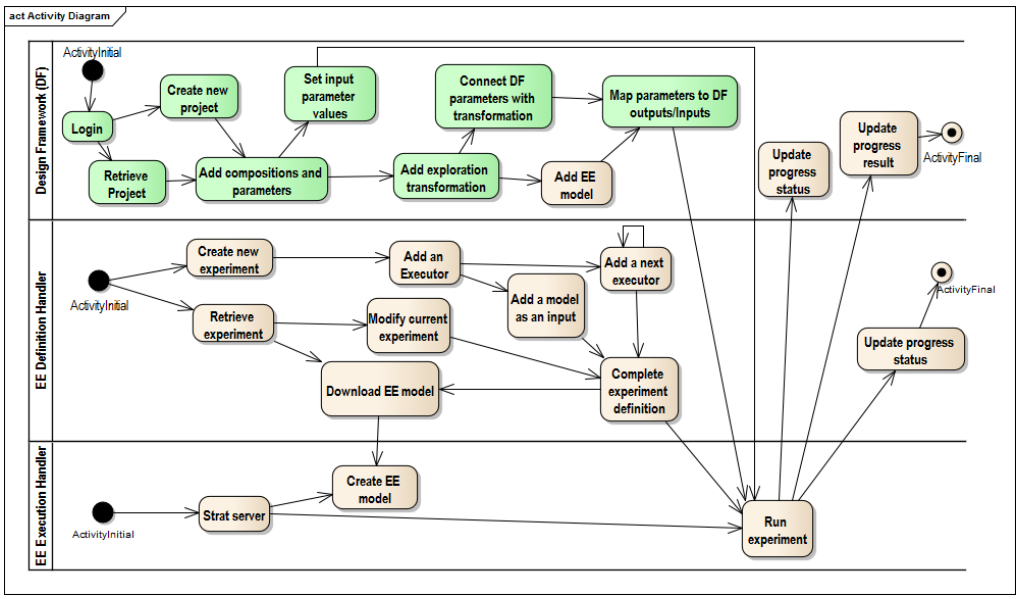


Figure 8 Activity Diagram of Main Action Flows

The activity diagram groups actions by the components. A block represents an action which is provided by one component. An arrow between two blocks points a flow direction. The arrows cross two different components indicate the interactions between two components.

Activities in DF

In DF, the actions in green have already been implemented. Only three actions are introduced in order to apply an experiment. It requires some preparations before add-

ing an EE model. For instance, signing into the DF system, creating or retrieving a project, adding compositions and parameters, adding transformations, connecting DF parameters with transformations. After these preparation steps, the user can add an EE model, map parameters to DF inputs/outputs and set input parameter values so as to run the relevant experiment in EE Execution Handler. The progress status can be updated during the execution. After completing the entire execution, the final results can be updated in DF.

Activities in EE Definition Handler

In EE Definition Handler, the user can create an experiment and define it by performing the following steps, shown in Table 9.

Table 9 Three Steps to Define an Experiment

Step	Description
Step 1	Add an executor.
Step 2	Select an executor, add a model as an input to the selected executor.
Step 3	Select an executor, add another executor as a successor to the selected executor.

The order of Step 2 and Step 3 can be reversed. Step 3 can be repeated many times until the experiment definition is completed. The user can also retrieve an existing experiment and modify it by starting from any step in Table 9 until the experiment definition is completed.

After completing an experiment definition, the user can run the experiment directly through the EE Definition Handler as well as download a relevant EE model. Running an experiment and creating an EE model are two actions provided by EE Execution Handler.

Activities in EE Execution Handler

In EE Execution Handler, a running server is a precondition for creating an EE model and running an experiment. An activity diagram for running an experiment is illustrated in Figure 9. In the activity diagram, *Update the execution progress status* action is provided by EE Definition Handler or DF. *Update the final result* action is provided by DF.

The system is waiting for a request from DF or EE Definition Handler. The request must contain an experiment ID to clarify which experiment needs to be conducted. After getting the request, the system starts to perform the required experiment by the following steps in Table 10.

Table 10 Six Steps to Run an Experiment

Step	Description
Step 1	Generate a unique run ID from the database and parse the experiment ID from the request.
Step 2	Check whether this request includes input parameter values. If it contains such information, continue with Step 3, otherwise, go straight to Step 4.
Step 3	Parse the parameter values and create a parameter-value file for each model before starting an execution.
Step 4	Get the next executor's information. For the first execution, the next executor refers to the first executor. If the next executor exists, continue with Step 5, otherwise go straight to Step 6.
Step 5	Create a runtime folder for this execution, set runtime input arguments and perform this execution. Update the progress status after this execution has completed. Go back to Step 4.
Step 6	Update the final results.

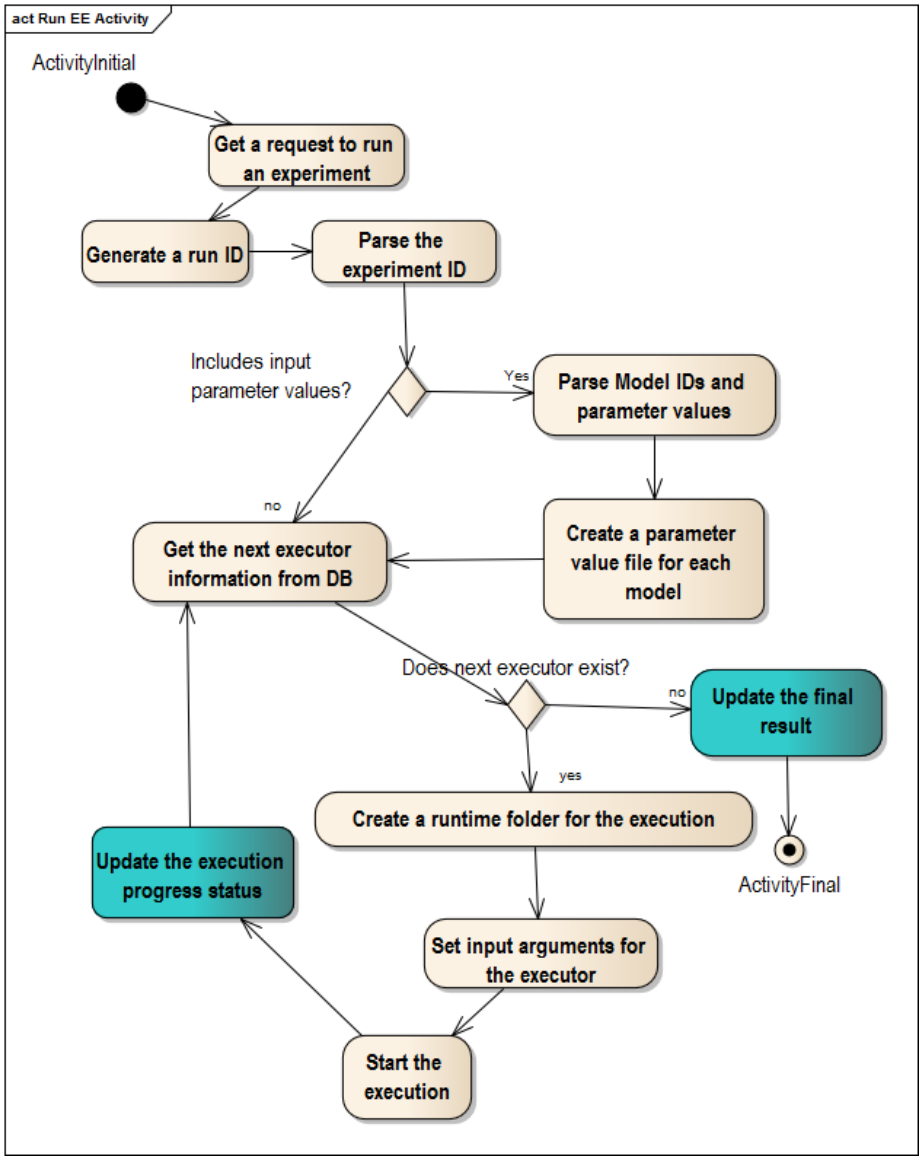


Figure 9 Activity Diagram for Running an Experiment

5.2.4. EE Physical View

After the detailed description of core components and interaction flows, this section discusses how to deploy the entire system physically.

In order to deploy the system, the EE part requires a web server, a database and an execution environment, while the DF part needs a web server and a database. The technologies applied for developing web servers and databases are the same. Depending on users' different needs, there are two ways to deploy the system. One is to deploy EE and DF together in one platform, shown in Figure 10. The other is to separate DF from EE, deploy it on another platform, shown in Figure 11.

The benefit of one platform is that the user can install the server package directly to his/her local device. There is no need for an internet connection to run the system.

One benefit of two platforms is from the developer's perspectives. Since both tools are still under development, the deployment of DF and EE on separate platforms can guarantee the separation of the development environments. Besides, applying a big powerful server on one separated platform for EE Execution Handler to deal with many heavy experiments allows DF continuing with the rest of tasks at the same

time. However, this deployment requires the internet connection between DF and EE Execution Handler.

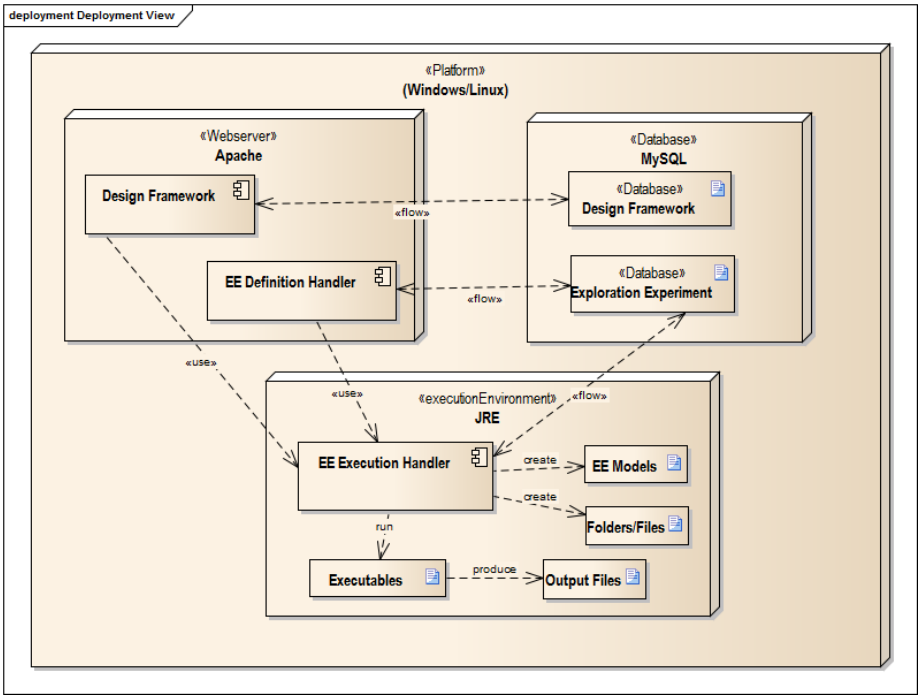


Figure 10 Deployment Diagram for One Platform

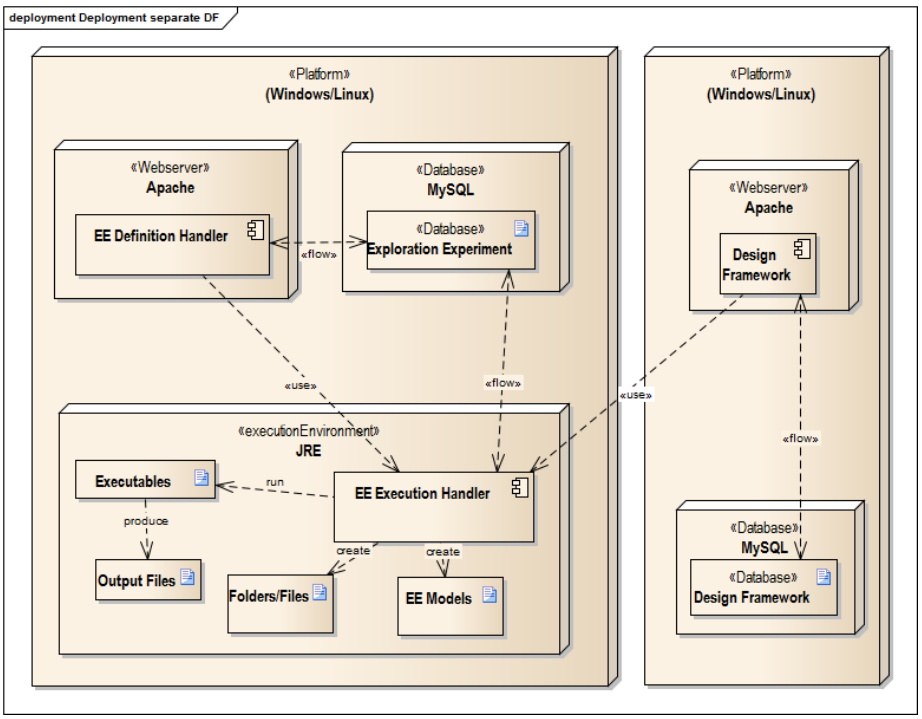


Figure 11 Deployment Diagram for Two Platforms

5.2.5. EE Logical View

From design to implementation, the EE Logical view explains more about how the EE components interact with each other in detail. Since the class diagram in EE Definition Handler part is very simple, this section only discusses the controller part of EE Executable Handler. A snippet of the relevant class diagram is shown in Figure 12.

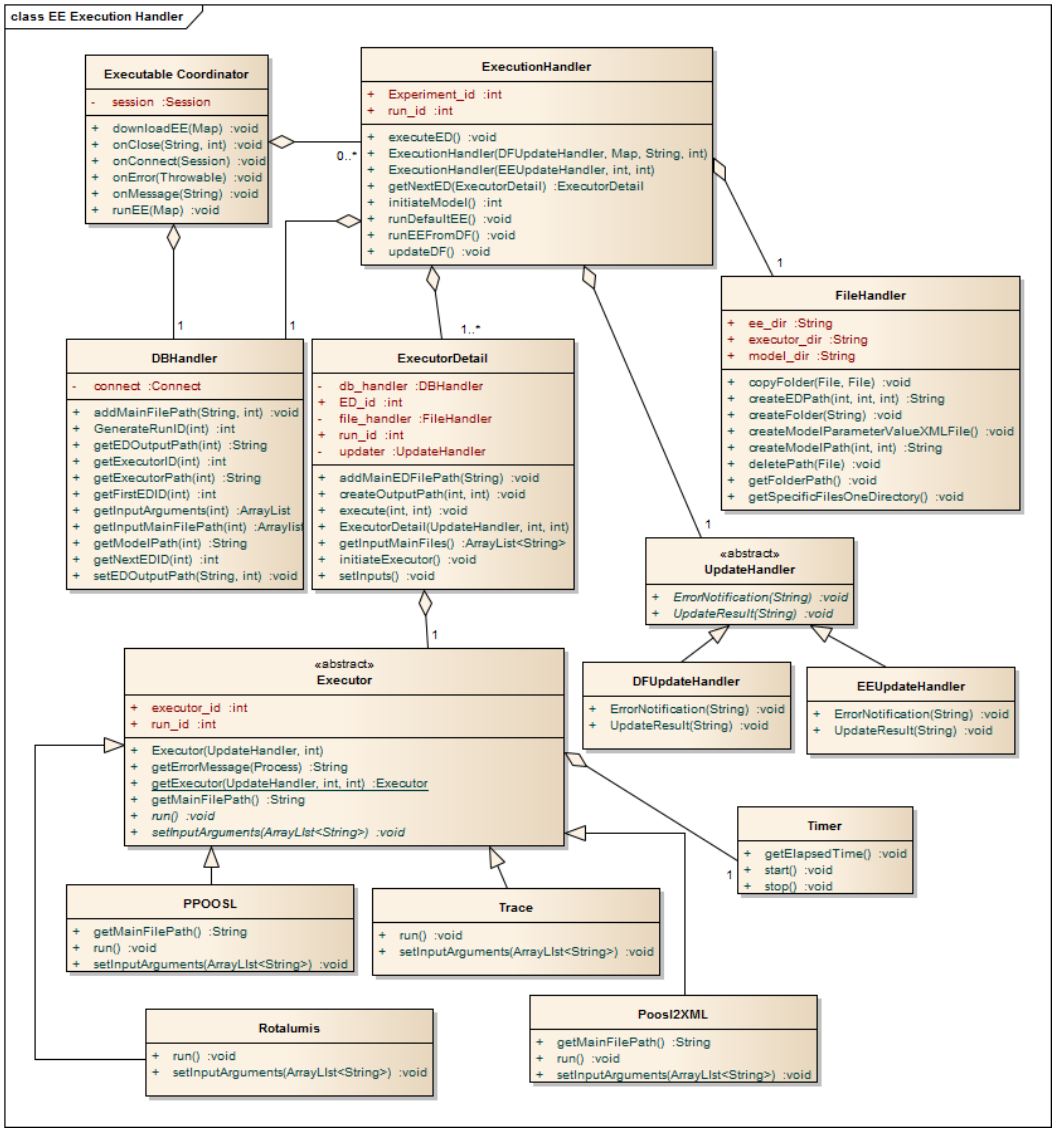


Figure 12 The Controller Class Diagram in EE Execution Handler

An *ExecutableCoordinator* handles various requests from DF and EE Definition Handler. It contains a list of *ExecutionHandler*, which is used to handle a single experiment execution chain. An *ExecutionHandler* contains a list of *ExecutionDetail*, which provides concrete functions about one executor's execution, such as creating output path in the runtime, setting the inputs for one execution. Every *ExecutionDetail* contains an abstract class *Executor*. As an abstract class, *Executor* is specialized by four concrete classes, *PPOOSL*, *Poosl2XML*, *Rotalumis*, *Trace*. Every subclass of *Executor* needs to implement two abstract functions: *setInputArguments* and *run* according to its own ways of setting input arguments and executing. If there is a main output file needs to be specified, the subclass also needs to override the function *getMainFilePath* from the *Executor*.

Besides *ExecutionHandler*, the other handlers are also used to complete an execution. *DBHander* mainly deals with database. *FileHander* is used to create output folders, copy/paste files to a certain directory and delete redundant runtime files. *UpdateHandler* is used to update results or progress status to DF or EE Definition Handler. It is an abstract class and specialized by *DFUpdateHandler* and *EEUpdateHandler* to update DF and EE Definition Handler in different ways. *Timer* is needed to get the execution time for an *Executor*.

5.3 TRACE Architecture

Apart from the EE architecture, the existing TRACE code refactor also involves architecture design.

5.3.1. TRACE Use Case Scenarios

TRACE is a visualization tool, which mainly generates two types of graphs:

- Gantt Charts (Single Gantt Chart, Gantt Chart Comparison among multiple files)
- DS Graphs (Radar Graph, Scatter Plot Graph 2D/3D, Parallel Coordinates Graph, Heat Graph 2D/3D)

Four user goals can be summarized:

- View a single Gantt Chart
- View a Gantt Chart comparison
- View a DS graph
- Check properties of a point/curve in a graph

The use case diagram is illustrated in Figure 13, and the use cases in green are existing use cases of visualizing a single Gantt Chart. Corresponding to the use case diagram, the use case scenario tables are also followed.

Use Case 1: View a Single Gantt Chart

Primary Actor: TRACE Tool User
Context of use: the user wants to view a Gantt Chart from a single file
Scope: Eclipse IDE or TRACE standalone application
Precondition: an Eclipse project needs to be opened in the project/package Explorer
Success Guarantees: the user opens a Gantt Chart file from a project and the system can display the Gantt Chart graph accordingly
Main Success Scenario
1: Tool User: Select a Gantt Chart file from a project in Project/Package Explorer
2: System: Display the Gantt Chart accordingly
3: System: Open the ESI Trace Property View
Extensions
2.a: Filter the claims or resources in the graph
2.b: Change the view type between Resource View and Activity View
2.c: Zoom or pan a Gantt Chart
2.d: Select colors for claims or resources
2.e: Select grouping criteria
2.f: Select dependency types

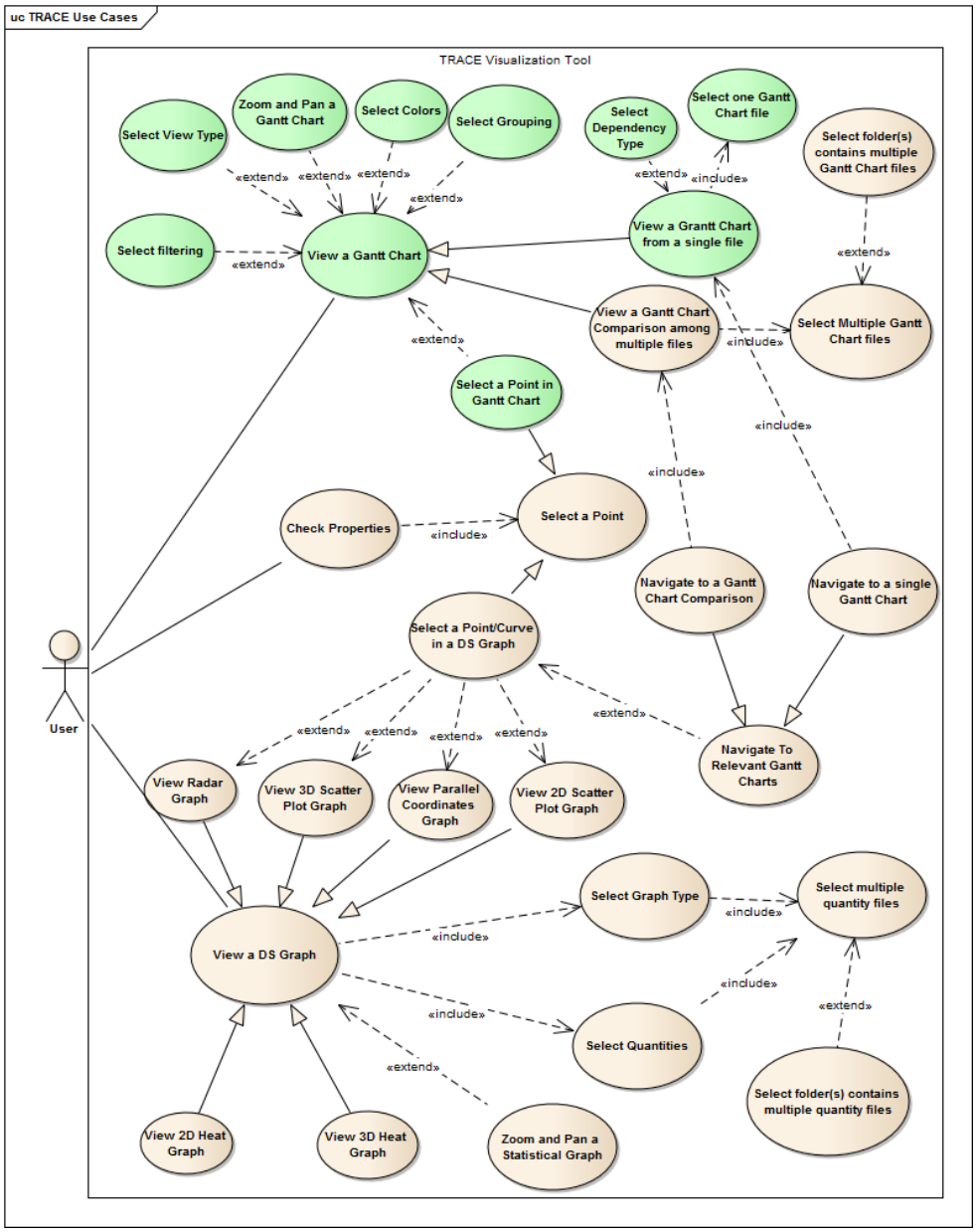


Figure 13 TRACE Use Case Diagram

Use Case 2: View a Gantt Chart Comparison

Primary Actor: TRACE Tool User
Context of use: the user wants to view a Gantt Chart comparison from at least two Gantt Chart files
Scope: Eclipse IDE or TRACE standalone application
Precondition: an Eclipse project needs to be opened in the project/package Explorer
Success Guarantees: the user opens multiple Gantt Chart files with Gantt Chart comparison option and the system displays the relevant Gantt Charts.
Main Success Scenario
1: Tool User: Select to open multiple Gantt Chart files
2: System: Display a Gantt Chart comparison graph accordingly
3: System: Open the ESI Trace Property View
Extensions
1.a: Select a folder or folders which contain multiple Gantt Chart files and open with comparison view
2.a -2.e are the same extensions as Use Case 1.

Use Case 3: View a DS Graph

Primary Actor: TRACE Tool User
Context of use: the user wants to view a DS graph from at least two quantity files
Scope: Eclipse IDE or TRACE standalone application
Precondition: an Eclipse project needs to be opened in the project/package Explorer
Success Guarantees: the user opens multiple quantity file with specific graph options and the system displays the relevant DS graph.
Main Success Scenario
1: Tool User: Select to open multiple quantity files
2: System: Display the graph configuration choices
3: Tool User: Select a graph type
4: System: Confirm the graph type and notify the limitation of quantity numbers
5: Tool User: Select the quantities
6: Tool User: Confirm to Generate the required DS graph
7: System: Display the required DS graph
8: System: Open the ESI Trace Property View and update the content
Extensions
1.a: Select a folder or folders which contain multiple quantity files and open with Design Space Visualization
7.a: Select a point in a DS graph, if the graph is of the type Rader Graph, Scatter Graph 2D/3D, Parallel Coordinates Graph
7.b: Zoom or Pan a DS graph

Use Case 4: Check the Properties in a graph

Primary Actor: TRACE Tool User
Context of use: the user wants to check the property of a point/curve in a graph
Scope: Eclipse IDE or TRACE standalone application
Precondition: a graph needs to be opened
Success Guarantees: the user select a point/curve in a graph and the system display all the relevant information on the ESI Trace View
Main Success Scenario
1: Tool User: Select a point/curve in a graph
2: System: Display all the relevant information of the particular point/curve on ESI Trace View
Extensions
1a: Navigate to related Gantt Charts, if the point/curve is in a DS graph

5.3.2. TRACE Development View

TRACE visualizes Gantt Charts and DS graphs regarding different input files. Therefore an organized pattern to handle input files, control data models, and present proper views is recommended. Besides, usability and extensibility are two important non-functional requirements for the TRACE tool. Taking those factors into account, the MVC architecture pattern is selected to develop the TRACE tool. The relevant pattern is illustrated in Figure 14.

Notice that the terminologies “Editor”, “Dialog”, “Wizard” and “Property View” are in the domain of Eclipse IDE. More details can be found on [13].

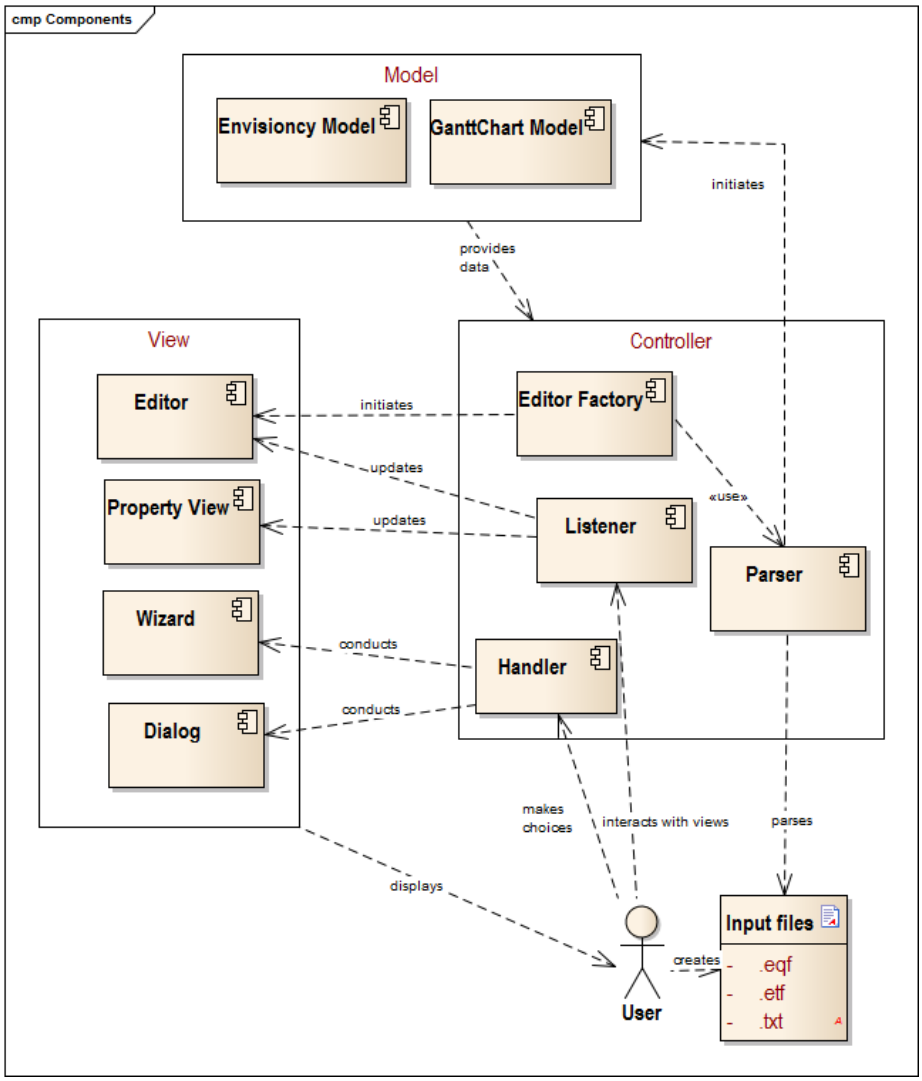


Figure 14 TRACE Component Diagram

Model

Unlike the models in EE architecture, which are concrete files and tables in local disks and database, the TRACE models are created at runtime. Two separate models are involved:

- Envisioncy Model: it contains the data structure of the DS graphs. All types of DS graphs use the same model as input, but display differently.
- GanttChart Model: it is an input for the Gantt Charts, which contains data structures.

View

The view displays editors, properties views, dialogs, and wizards to the user. According to different representation types, it is divided into four parts:

- Editor: it organizes how graphs can be displayed in the editor area. Depending on different model types, it also contains two editor types: Envisioncy Editor and Gantt Chart Editor.
- Dialog: when the user needs to specify some particular options in order to proceed to the next step some popup dialogs are required to guide the user's choices.
- Wizard: it is similar to the dialogs, but multiple pages are involved. It displays the options for the user and helps to make choices until it reaches the final step.

- Property View: it displays the properties of the selected points/curves in a graph.

Controller

The controller parses input files, creates models, handles user's interactions, conducts procedures, and updates views. Regarding its particular responsibilities, it is grouped into four parts:

- Editor Factory: the factory which initiates an editor and fills the content of the frame.
- Handler: it handles the user actions for dialogs and wizards, and takes user's choices as an input to either manipulate models or update views.
- Listener: it registers all the event listeners in the graphs and the property view. When a user interacts with graphs or the property view, it will update the view's representation.
- Parser: it parses files from the directory and initializes models.

5.3.3. TRACE Logical View

After the introduction to all the TRACE components, the TRACE logical view explains more about how its components are organized internally and how they interact with each other.

Figure 15 is a snippet from the MVC architecture, and illustrates the interaction among the editors, editor factories and models.

Every editor is a specialized Eclipse *EditorPart*. Considering different input file formats, the *GanttChartEditor* takes care of Gantt Chart files and *EnvisioncyEditor* is used to represent quantity files. There are two ways to produce Gantt Charts. A *SingleGanttChartEditor* is used for displaying a single file, while a *MultipleGanttChartEditor* is used for displaying multiple file comparison. An editor contains a frame, which is used to display a required graph. Each editor is responsible for its own layout and the frame initialization.

A concrete editor contains an editor factory, which is responsible for the content of the frame. A factory refers to a unique Project, which consists of all the Gantt Chart models and Envisioncy models. By specifying which model, the editor factory completes the content of the relevant frame.

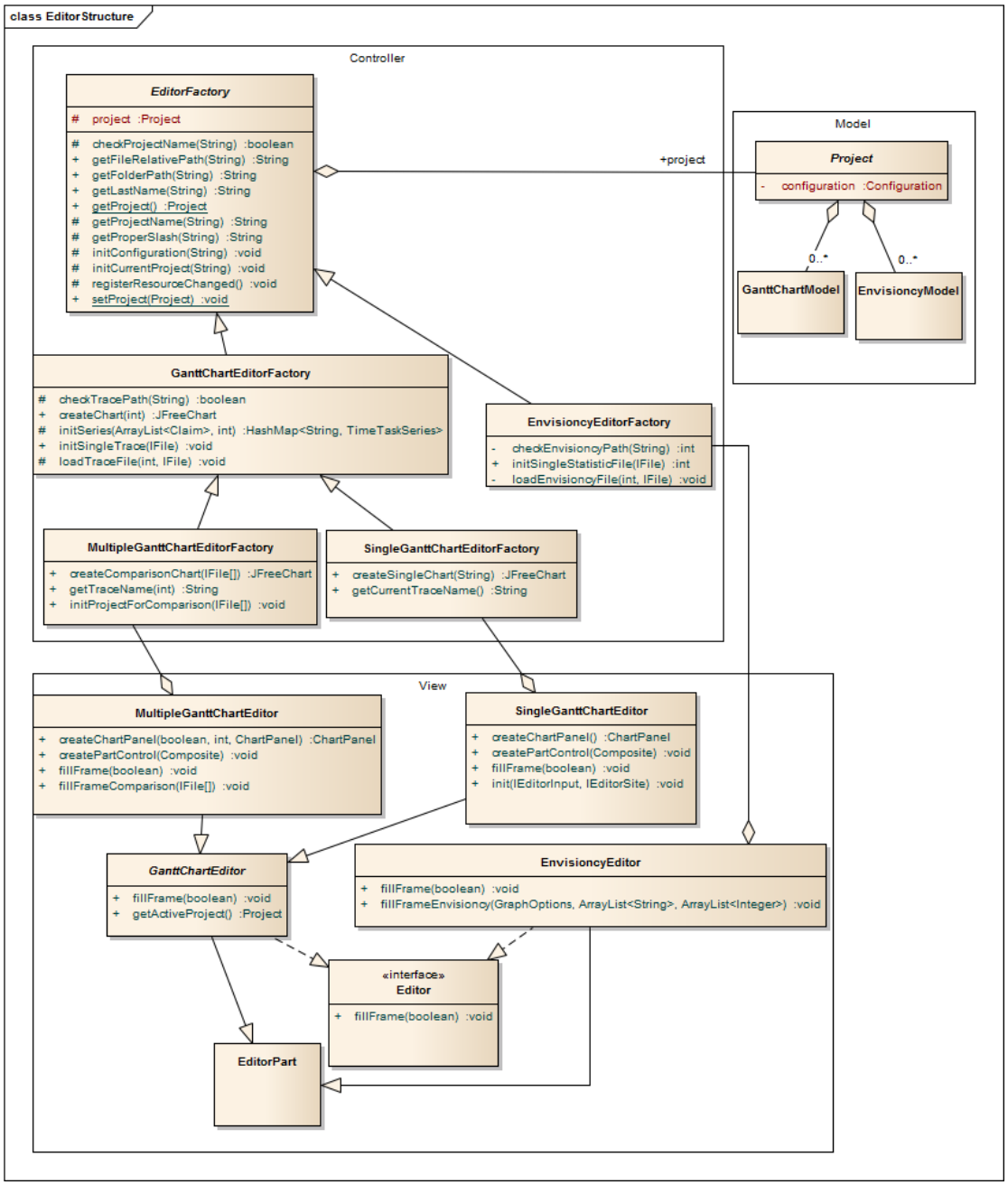


Figure 15 TRACE Editor Class Diagram

6.Design & Implementation

The previous chapter provided a high-level look at the system and its intended architecture. In this chapter, a close look at the system is taken. Many design alternatives are involved in both EE development and TRACE extension development. Based on both functional and non-functional requirements, the major design issues are discussed in this chapter.

6.1 Introduction

After describing the overall architecture of both EE and TRACE parts, the concrete design and implementation are discussed in this chapter. Since development of EE includes two persons' effort, technology alternative evaluation parts still involve some joint effort of two persons. However, the implementation tasks are completely separated. The rest of the chapter describes the main design issues involved in both EE and TRACE parts.

6.2 EE Design and Implementation

Design and implementation of EE Definition Handler, EE Execution Handler and the interfaces in-between are the main parts of EE development. There are many technological options during the implementation. Selection of the proper technologies and application of these technologies to the overall architecture are essential issues.

6.2.1. Individual Scope

My individual scope for EE implementation comprises the following:

- Work under EE Definition Handler part
  The main effort is on the GUI and GUI Handler parts, which handles user interaction with GUI, such as defining an experiment from a GUI, uploading a local model to the server, updating a runtime experiment status.
- Work under EE Execution Handler part
  The main effort is on the Executable Coordinator, Execution Handler, Update Handler parts and interfaces between EE Execution Handler and DF. The major task is executing a defined experiment, which involves taking commands from DF or EE Definition Handler, parsing the input stream, running the execution chain, updating the progress results back to DF or EE Definition Handler.

The components mentioned above refer to the EE component diagram in Figure 7.

6.2.2. EE Definition Handler

As a web application, there are several design or technology issues involved.

Server Language

It is important to select a server language for a web application first. After a preselection among all the alternatives, two server languages, PHP and JSP, are considered. Table 11 shows the detailed comparison between PHP and JSP.

Table 11 Server Language Alternative Analysis

Criteria	PHP	JSP
Usage	Popular. There are lots of available solutions available online.	Less popular.

Maintainability	PHP5 become object oriented. It is a new feature in PHP5, so it is not mature enough.	JSP uses Java programming language, which is completely object oriented.
Development Environment	No good development environment for debugging.	Many good Java development environments are avaiable, such as Eclipse, Netbeans.
Learning Curve	Easy to learn.	Difficult to learn.
Performance	Slower.	Faster. However if a developer is not familiar with it, the performance can be slow.
Security	Less Secure. The secure functionalities need to be added by developers.	More secure. JSP as java in general is based on a secure infrastructure.

Since this project aims for a prototype, security and performance are not the main concerns. The feasibility of all the functional requirements is most important. Compared with JSP, PHP is easy to start up and contains sufficient online support. Besides, the current DF server language is PHP, so it is a better choice for this project.

Integrated Development Environment

After selecting the server language, WampServer is applied as a web development environment. It provides a good IDE for an Apache server, PHP and a MySQL database. See more details in [8].

Server Development Framework

In order to apply MVC architecture pattern better, a simple framework PHP-MVC is applied. The old way of using PHP is to send the user from *index.php* to some other PHP files. PHP-MVC organizes all the PHP files into three parts: model, view and controller.

- Model deals with MySQL database.
- View specifies the outlook of the webpages.
- Controller contains the functionalities of interacting with the database and updating the webpages.

One special feature of PHP-MVC is the way it invokes the functionalities. Instead of importing a class and calling the functionality that belongs to that class, all the functionalities are invoked through URLs. The URL looks like:

http://mainurl/controller/action/first_parameter

- *controller* refers to a PHP class.
- *action* refers to a function belongs to the *controller* class
- *first_parameter* refers to the first parameter of the *action* function

If there are more than one parameters, all the parameters can be attached to the URL in order.

More details and a bare-bone structure of PHP-MVC are available on [9].

Graphical User Interface

In order to provide a better way of defining an experiment, a GUI is introduced. The technology to develop a GUI is HTML5 canvas, because it is simple and provides enough features for the prototype. Besides HTML5 canvas, some JavaScript files are needed to interact with the canvas.

6.2.3. EE Execution Handler

Design and implementation in EE Execution Handler also involve several issues.

Programming Language

Java is selected as a programming language for several reasons:

- Sufficient libraries and frameworks to support communication.
- A well-structured object-oriented language, which is easy to implement.
- Most of executables are JAR files, which are implemented in Java.

Conducting Executable Alternatives

When a particular executable needs to be executed in EE Execution Handler, it can be either executed through the system call or use the web service. Table 12 is a comparison table of the two alternatives.

Table 12 Conducting Executable Alternative Analysis

Criteria	Web Service	System Call
Deployment (Shown in Figure 16)	Distributed. Each executable can be deployed in any other platform.	Centralized. All the executables need to be deployed in the same platform with EE Execution Handler.
Implementation	Needs a framework to support web service.	Only needs to invoke a system call from Java.
Network Dependency	The network is needed for communication.	All the executions are in the local environment.
Extensibility	Extensions can be done independently. Only requires to report a valid URL to EE Execution Handler.	Extensions need to be integrated with the existing EE Execution Handler.
Performance	The network traffic can influence the response time.	Needs to concern resource usage overload when multiple experiments are conducted at the same time.

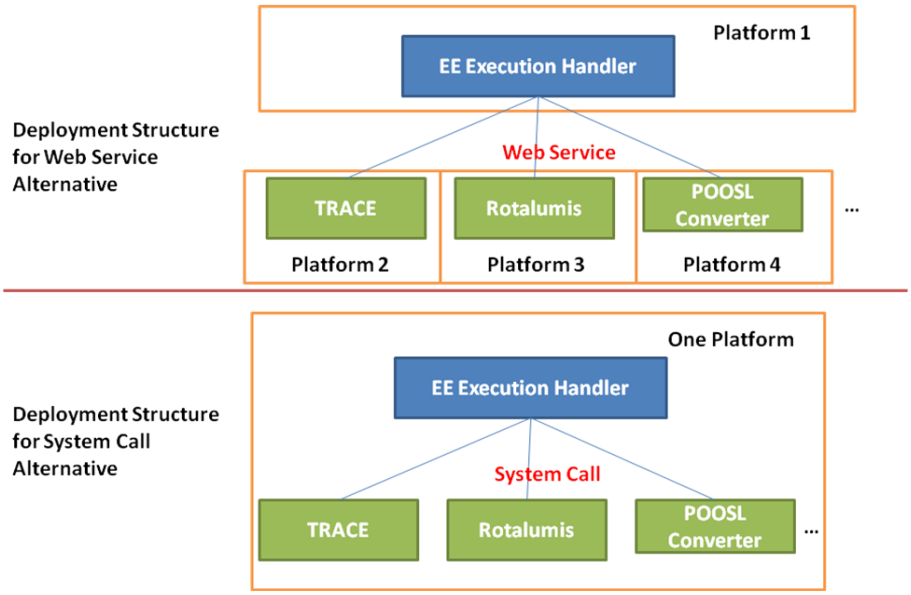


Figure 16 Deployment Structures for Web Service and System Call

From the above analysis, both alternatives have their strength and weakness. Due to the time frame, the system call alternative is selected for the following reasons:

- It is good enough to invoke all the executables in this project.
- Use this EE Execution Handler locally is preferable for the user, since web services depend on the web server and the network connection.

- For a demonstration, the number of experiment executions in parallel is no more than ten. Therefore, the resource overload is not a concern at the moment.

6.2.4. Interfaces

Runtime Functional Calls

There are six runtime functional calls involved between DF, EE Execution Handler and EE Definition, shown in Figure 6. The runtime communication among them can be supported by many technologies. Two alternatives, web socket and web service, are considered. The analysis from different aspects is listed in Table 13.

Table 13 Runtime Functional Call Alternative Analysis

Criteria	Web Socket	Web Service
Communication Direction	Bi-directional. Both a client and a server can start to talk independently once the connection is established.	Single-directional. A client needs to use polling to get the response, or the client also provides web services URL for calling back.
Base Technology	TCP	HTTP
Implementation	More complex. Have to implement everything on top of web socket protocol. However there is existing frameworks to use, such as Jetty in Java.	Simpler. Use URLs and wrap existing functionalities.
Performance	Low latency.	High latency.
Network Traffic	Once a connection is established, only necessary messages are transferred.	If a client needs response from a server, there are many redundant messages due to the polling mechanism.

EE Execution Handler mainly handles execution requests, conducts concrete experiments and sends results back to initiators. The execution time for experiments varies a lot. Some may only take a few seconds, on the contrast, some may take hours or days. For the executions that require very short time, there are no big differences by using web socket or web service. However, for the executions that require hours or days, the web service technology using polling mechanism costs a lot of redundant messages. It may overload the network. Since EE Execution Handler may send some notifications back to DF or EE Definition Handler at any time during an execution, a bi-directional communication is preferable.

As a result, web socket, which can provide bi-directional communication with less network traffic and better performance, is selected to implement the runtime functional call interfaces. The existing framework, Jetty, is applied to set up the server of EE Execution Handler.

DF Integration Limitation

There is an issue at the DF Apache server side. The server terminates all the live sessions every minute. If an execution at EE Execution Handler takes more than one minute, the established connection is not alive any longer. As a result, DF cannot get any response from EE Execution Handler after one minute. The web socket way of communication can only start a connection but cannot send the response.

In order to solve it, web service technology is brought. DF provides several callback URLs for EE Execution Handler. When executions finish, EE Execution Handler can invoke a relevant callback URL to update the results. Therefore, the way of runtime

communication between DF and EE Execution Handler is a combination between web socket and web service. DF starts a request using web socket connection and EE Execution Handler updates the results using web service.

File Transmission

During the execution, the executables can produce many output files. Some of output files need to be transmitted to DF. Depends on the deployment view, two ways are applied to accomplish the file transmission.

- Local Copy Paste - if all the components are on one platform, which refers to Figure 10, all the files are physically in the same machine. Local copy paste is the simplest method to transmit files.
- File Transfer Protocol (FTP) - it is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the Internet.[10] If DF and EE Execution Handler are deployed on two platforms, which refers to in Figure 11, the FTP method is used. The FTP server is at the EE Execution Handler side, and client is at the DF side.

6.3 TRACE Design and Implementation

TRACE extension development does not involve many technology alternatives, since it already limited to the Eclipse plug-in development environment. However, it still contains some design issues during the implementation.

6.3.1. TRACE Standalone Alternatives

Due to the complexity of Eclipse, a standalone version of TRACE is requested to extract the TRACE functionalities. At the same time, the Eclipse plug-in still needs to be kept. Therefore, both the Eclipse plug-in and the standalone application need to support the same functionalities. There are two alternatives for the TRACE standalone application. One is an Eclipse product using Rich Client Platform (RCP) technology. The other is a pure Java application. Table 14 illustrates the analysis between the two alternatives.

Table 14 TRACE Standalone Alternative Analysis

Criteria	Eclipse Product (RCP)	Java Application
Maintainability	The source code can support both the Eclipse Plug-in and the standalone version.	Two different source codes are required to support the Eclipse plug-in and the Java standalone application separately.
Library Support	It can use Eclipse features to build UI extensions.	It can use Java library to build its own UI.
Technical References	Some online tutorials are available.	Have similar source code for UI features of viewers (e.g., Envisioncy, Resvis)
Size of Application	Around 40 MB.	Can be smaller than 1MB.
Complexity of Developing new features	Main challenge is to transfer Envisioncy code into Eclipse plug-in feature.	Main challenge is to transfer the current TRACE eclipse plug-in to Java features. Envisioncy code still needs to be merged to the Eclipse plug-in.
Usability	Exactly the same user experience of the Eclipse plug-in.	User interaction may be different, due to a different UI.
Redundant features	It inherits lots of Eclipse basic features.	It only keeps the necessary features.

The biggest advantage for an Eclipse product is maintainability, because the same code is used for the Eclipse plug-in and the standalone application. The RCP technology only builds a simplified Eclipse framework for the standalone application. The weakest part of an Eclipse product is that some redundant features from Eclipse core framework cannot be removed. Although it is much more simplified than Eclipse, it is around 40 MB.

As opposed to an Eclipse product, the most attractive criterion of a Java application must be its footprint. However, two copies of code need to be implemented, due to the requirements. The workload of extending new TRACE features is doubled as well as the effort of maintaining both copies of code.

Taking more maintainability and complexity of developing new features into account, an Eclipse product is selected to build a TRACE standalone application.

6.3.2. File Structure for Quantity Attribute Value

Quantity attributes are introduced into TRACE while merging Envisiongy functionalities. These attributes represent some statistical values for a single trace file or more trace files. By loading data of quantity attributes, some DS graphs from Envisiongy can be generated. The definitions of the quantity attributes are located in the configuration file. However, the way to store the quantity values, which are related to one or more trace files, can be organized differently.

In general, there are three alternatives to organize the quantity values.

- **Alternative One:** create a separate quantity file with the ".eqf" extension to store the quantity values as well as relevant links to a configuration file and its relevant Gantt Chart files.
- **Alternative Two:** attach quantity values to existing Gantt Chart files, which end with the ".etf" extension. Distinguish quantity values by beginning with "Q".
- **Alternative Three:** use a separate quantity file as Alternative One, as well as attach quantity values to existing Gantt Chart files as Alternative Two.

Table 15 shows the evaluation among these three alternatives.

Table 15 File Structure for Quantity Attribute Value Alternative Analysis

Criteria	Alternative One	Alternative Two	Alternative Three
Use Case Coverage (It is related to one single Gantt Chart only or multiple Gantt Charts)	Can support both. It only needs one link to a list of Gantt Charts.	Only can support a single Gantt Chart, since one ".etf" file can only describe one single Gantt Chart.	Can support both. If it is related to a single Gantt Chart, the quantity value can be attached to the related ".etf" file, or declared in a ".eqf" file with a link to the Gantt Chart file. If it is related to multiple Gantt Charts, it has to use a ".eqf" file to store a list of Gantt Charts.
Usability	Select ".eqf" files to display graph.	Select ".etf" files to display graph.	Select ".etf" files or ".eqf" files to display graph.
Complexity	One type of file to parse.	One type of file to parse.	Two types of files need to parse.
Performance	It is good in general, because a ".eqf" file only contains several lines to describe quantity values.	It depends on the size of ".etf". The bigger size, the more time it takes to parse.	The same situation. Parse ".eqf" file is very fast and parse ".etf" file depends on how large the file is.

Since performance is a very important non-functional requirement, the response time is too slow when loading large ".etf" files. The alternatives which include loading quantity value from ".etf" files are excluded. Then the Alternative One, a separate ".eqf" file structure, is chosen to organize quantity attribute values.

7.Verification & Validation

To ensure that the implemented software fulfills the intended functionality and behavior that was described in the previous chapters, verification and validation are applied to both EE and TRACE parts. This chapter describes various test cases for verification and a case study for validation.

7.1 Introduction

According to the requirements which are discussed in Chapter 4, this chapter provides sufficient test cases and a concrete case study to ensure EE and TRACE perform well and also meets customer's needs. For verification of EE and TRACE, test cases are used to check all the functional requirements and some evaluations are performed to evaluate non-functional requirements. A non-confidential case study, which is provided by TNO-ESI, is used for EE validation. The simulation results from this case study are applied for TRACE validation. Furthermore, the latest TRACE tool has already released on the official website.[7] TNO-ESI approved the latest TRACE tool before the release.

7.2 EE Verification

7.2.1. Functional Test

The functional tests in EE are conducted manually through the user interfaces of EE and DF. The concrete test cases are shown in Table 16 and Table 17.[2] For the same functionality, both acceptance test cases and exception test cases are conducted. The "Req. ID" in the test case tables refers to the IDs in the requirement tables in Section 4.1.1.

Table 16 Test Cases – EE Definition Handler

ID	Functionality	User Activity	Req. ID	Expected Result	Test Result
A1	Create an experiment	Enter a name.	A1	Create an experiment successfully.	Passed
A2		No name is entered.	A1	Show no name notification.	Passed
B1	Retrieve an executor	Select an executor from the combo box.	A2	The selected executor is added to the canvas.	Passed
B2		Retrieve an executor without getting an experiment first.	A2	Show no experiment notification.	Passed
C1	Upload a model	Select a zip file with maximum 8MB, enter main file path and model name.	A3	Upload model successfully.	Passed
C2		Select a non-zip file.	A3	Show non-zip file notifica-	Passed

[2] Test case ID for the same functionality begins with the same letter. For instance, "A1" and "A2" test the same "Create an EE" functionality with different user activities.

				tion.	
C3		Select a model which exceeds 8MB.	A3	Show exceeding boundary notification.	Passed
C4		Upload without a file attached.	A3	Show no sufficient information notification.	Passed
C5		Upload without entering a main file path or a model name.	A3	Show no sufficient information notification.	Passed
D1	Retrieve a model	Select a model from the combo box.	A4	Add the selected model to the canvas	Passed
D2		Neither retrieve a model nor upload a model, but click “OK”.	A4	Show no model notification.	Passed
E1	Add a connection from a model to an executor	Select an executor and right click to add a model.	A5	Connect a model to the selected executor.	Passed
F1	Add a connection from an executor to another executor	Select an executor and right click to connect to the next executor.	A6	Connect an executor to the selected executor as a successor.	Passed
G1	Download EE model	Download an EE model for the current experiment.	A7, A14, A15	Get a ".ee" model.	Passed
G2		No experiment is selected.	A7	Show no experiment notification.	Passed
G3		No model is defined in the current experiment.	A7	Show no model notification.	Passed
H1	Retrieve an experiment	Select an experiment from the combo box.	B1	Display its execution flow on canvas.	Passed
I1	Run an experiment with default values	Current experiment exists and run the current experiment.	A8, A10, A11, A12, A13, B2, B3, B4	The experiment is executed and the runtime status is updated on the UI.	Passed
I2		No experiment is selected.	B2	Show no experiment notification.	Passed
I3		No model is defined in the current experiment.	A8, B2, C1, C2	Show no model notification.	Passed
J1	Display progress status or error notifications	Start an experiment successfully	C1, C2, C3	Show the progress status of the running experiment.	Passed

Table 17 Test Cases - DF

ID	Name	User Activity	Req. ID	Expected Result	Test Result
K1	Upload an EE model	Add a new transformation Select a well-defined EE model.	A17, A18	The EE model is added and its relevant parameters are extracted to DF.	Passed
K2		There is syntax error inside an EE model, in case that an EE model is modified by the user.	A17, A18	Show error notification.	Passed
L1	Set input values	Enter proper input values.	A19	Trigger the experiment successfully.	Passed
L2		Input values are incorrect for the model.	A19, C3, C4	Trigger the experiment but show error messages.	Passed
M1	Start an experiment	Whenever something changes in the parameter values or mapping, it will trigger an execution.	A8, A9, A10, A11, A12, A13, A20, B4, C2, C4	Trigger the experiment successfully and update the progress status.	Passed
M2		Not all the input values are set.	A8, A20	Experiment is not executed.	Passed
N1	Display output results	Start an experiment successfully and the execution flow is completed.	A16, A21	Show the final results on the DF UI.	Passed
O1	Display error notifications	Start an experiment successfully and error occurs during the execution.	C3, C4	Show error notification on DF UI.	Passed

From above tables, there are 15 groups with 29 test cases in total. The correctness of all the EE functional requirements are passed.

7.2.2. Non-Functional Requirements Evaluation

Non-functional requirements are also important during the EE design and implementation. The four non-functional requirements of EE have already discussed in Chapter 4. The evaluation results are described as follows:

- Loose-coupling
  From DF's perspective, it only gets the necessary information from EE. The runtime interfaces between DF and EE Execution Handler are only three functional calls to execute an experiment, update final results, and update progress status. It keeps all the execution details apart from DF. Furthermore, an EE model, as a file interface between DF and EE, only contains the information of experiment ID, experiment name, input parameter and output parameter. The input parameter information is from the external model, like POOSL model. The output parameter information is from the last executor of an execution chain. All the intermediate executors' information is not mentioned in an EE model.

  The transmission mechanism of the framework to upload an external model to a DF can not only parse EE models but also handle Excel and Matlab

models. Moreover, the way to execute an EE experiment or other experiments like Excel experiment are exactly the same. Introducing EE to DF does not change any existing DF features.

From EE's perspective, EE is independent from DF. It has its own user interface which is mainly used for defining an experiment. Furthermore, the user can also execute an experiment from EE Definition Handler.

- Generality
  There is no restricts to model type while uploading a model from the EE UI. However, in order to generate an EE model, input parameter information needs to be extracted from the uploaded model. One executable file should be added to EE to extract parameters from the model. In order to execute an experiment with a set of parameter values, another executable file is also needed to set parameter values to the model. When a new model type is introduced, the developer needs to wrap two executables for extracting parameters and setting parameter values. One executable that can provide both functionalities is also suitable. By introducing executable(s) that can deal with model parameters, a new model can be used in EE. The generality of models meets the requirements.

  Referring to the class diagram of the EE Execution Handler in Figure 12, every concrete executor extends abstract class *Executor*. All of the executables are executed by invoking the same *run* method externally. The generality of the executables also satisfies the requirement.

- Usability
  The EE UI is very simple and easy to use. It is measured by the following aspects:
  - Exception Handler: many exception handlers are implemented to handle unexpected user interactions. The relevant test cases are shown in Table 16.
  - Canvas: with a canvas to define an experiment, it provides a clear execution flow.
  - Similar user experience with DF: the way of displaying a popup dialog to add executors and models are the same as DF. It provides a similar user experience for DF users.
  - Update status: the progress status of a running experiment is updated on the UI. It provides an overview of an entire execution chain in runtime for the user.

- Extensibility
  Only three steps are needed when a new executable is added:
  - Step 1: store the executable file at the server.
  - Step 2: add executable name and full path to the EE database.
  - Step 3: create a new Java class extending the abstract class *Executor* and implement the two abstract methods *setInputArguments* and *run*. If a main file is generated during the execution override the methods *getMainFilePath*.

  During the EE implementation, the above steps were applied when "parameterizedpoosl.jar", "poosl2xml.jar", "rotalumis.exe", and "trace.jar" were added.

7.3 TRACE Verification

7.3.1. Functional Test

The functional tests in TRACE are conducted manually through the user interfaces of both Eclipse and the standalone application. The concrete test cases are shown in the

tables below. Every test case is related to one or more functional requirements described in Section 4.2.1.

Table 18 Test Case - Gantt Chart Comparison

ID	Functionality	User Activity	Req. ID	Expected Result	Test Result
A1	Open a Gantt Chart comparison graph	Select multiple ".etf" files from the Project Explorer and compare with "Trace Comparison".	A24, A25, A26	Display a Gantt Chart in one editor and sort all the claims by name.	Passed
A2		Select a folder or folders which contain multiple ".etf" files and compare with "Trace Comparison".	A24, A25, A26		Passed
A3		Select fewer than ".etf" files to compare.	A24	Show no files notification.	Passed
B1	Gantt Chart Comparison functionalities	Zooming	A27	Perform the selected functionalities successfully.	Passed
B2		Panning	A27		Passed
B3		Filtering the attributes	A27		Passed
B4		Grouping	A27		Passed
B5		Coloring	A27		Passed
C1	Display Gantt Chart properties	Click a point from any claim of a Gantt Chart.	A28	Show the properties in a "ESI Trace Properties" view.	Passed
D1	Single Gantt Chart existing functionalities	Opening a Gantt Chart, zooming, panning, selecting a view type, coloring, grouping, filtering, showing dependencies, exporting charts	A29	Perform the existing functionalities successfully.	Passed

Table 19 Test Case - DS Graph

ID	Functionality	User Activity	Req. ID	Expected Result	Test Result
E1	Open a DS graph selection dialog	Select multiple ".eqf" files from the Project Explorer and select "Design Space Visualization".	A30	Display the DS graph selection dialog successfully.	Passed
E2		Select a folder or folders which contain multiple ".eqf" files and select "Design Space Visualization".	A30		Passed
E3		Select fewer than ".eqf" files.	A30	Show no files notification.	Passed
F1	Select DS graph type and quantities to display.	Select a DS graph type and the suitable number of quantities.	A31, A32	Perform the selected functionalities successfully.	Passed
F2		Select a DS graph type with unsuitable number of quantities.	A31	Show unsuitable quantity number notification.	Passed

G1	Display DS properties	Display a non-heat graph and select a point/curve from the graph.	A33	Show the properties in a "ESI Trace Properties" view.	Passed
H1	Display a navigation dialog	Display a non-heat graph and "Double Click" a point/curve, which is related to multiple ".etf" files.	B7	Display a navigation dialog successfully.	Passed
H2		"Double click" a point/curve's property through "ESI Trace Properties".	B7		Passed
I1	Navigate through a DS point/curve to a single Gantt Chart	Display a non-heat graph and "Double Click" the point/curve, which is related to one ".etf" file.	B6	Display the relevant single Gantt Chart successfully.	Passed
I2		"Double click" a point/curve's property through "ESI Trace Properties".	B6		Passed
I3		Select a single Gantt Chart from the navigation dialog.	B6, B7		Passed
I4		No file is selected from the navigation dialog.	B6, B7	Show no file notification	Passed
J1	Navigate through a DS point/curve to a Gantt Chart comparison	Select multiple files from navigation dialog and display Gantt Chart comparison.	B6, B7	Display the relevant Gantt Chart comparison successfully.	Passed
J2		Fewer than two files are selected to compare.	B6, B7	Show incorrect file number notification.	Passed

Table 20 Test Case - TRACE Executable File

ID	Functionality	User Activity	Req. ID	Expected Result	Test Result
K1	Execute the TRACE executable file	Set the correct input arguments and start the execution from command line.	A34, A35, A36	Execute the TRACE ".jar" file and generate Gantt Charts.	Passed
K2		Set the incorrect input arguments and start the execution	A34, A35	Show error notification	Passed

Table 21 Test Case - TRACE Standalone Application

ID	Functionality	User Activity	Req. ID	Expected Result	Test Result
L1	Support the same functionalities as	Conduct all the test cases in Table 18 and Table 19.	A22	Get the same results as Eclipse plug-	Passed

	Eclipse plug-in version.			in version.	
M1	Work independently	Uninstall the Eclipse IDE and only open the standalone version.	A23	Work without Eclipse IDE successfully.	Passed
N1	Support multiple operating systems	Install the standalone version on Windows, Linux and Mac OS and test it.	B5	Work properly on the tested platforms.	Passed

In above four tables, there are 14 groups with 29 test cases in total. The correctness of all the TRACE functional requirements are passed.

7.3.2. Non-Functional Requirements Evaluation

In Chapter 4, three non-functional requirements are discussed for TRACE development. The evaluation results are described as follows:

- Usability
  As a main non-functional requirement, many methods are performed to provide a better user interface.
  - Introduce two icons (and) to distinguish Gantt Chart files and quantity files.
  - Organize all the files in a flexible way. The user can either put all the files in one folder or separate them in a number of subfolders.
  - Provide sufficient exception handlers to give notifications, when the user makes some mistakes. For example, select only one file and open a Gantt Chart comparison. All the relevant test cases are in Table 18 and Table 19.
  - Provide more than one way to realize the same feature. For example, the user can navigate from a DS graph to a Gantt Chart by interacting with the DS graph or with the "ESI Trace Properties" view.

- Performance
  Gantt Chart files and quantity files need to share one configuration file, therefore, loading configuration file once at the first time is enough. In the old TRACE tool, a configuration file needs to be loaded whenever open a Gantt Chart file, even the same configuration file has already been loaded. In the new tool, opening the first Gantt Chart file requires the same time as the old tool. Afterwards, opening any other Gantt Chart file is on average three times faster.

- Extensibility
  New "Eclipse Extension point" features, like new buttons on the toolbar, are completely independent to the existing framework due to the way of plug-in tool development.

  For architectural features, like a introduction of another type of graph, the developer requires a deep understanding of the TRACE architecture. The TRACE code is implemented according to the MVC pattern, see Figure 15. If a new graph needs to be added, only three classes need to be added basically. One specialized *EditoryFactory* needs to be introduced to the controller part, one specialized *Editor* is required to the view part and a new model needs to be added to the model part.

7.4 EE Validation

A concrete case study is used in this section and shows how EE helps to perform a Design-Space Exploration for a multiprocessor system. Figure 18 to Figure 20 are screenshots from the EE tool and illustrate its user interface.

7.4.1. Case Study Introduction

A multiprocessor system consists of a (set of) parallelized application(s) that are mapped onto the multiprocessor platform. Figure 17 illustrates a task graph of the parallelized application and the battery-powered multi-processor platform. By mapping tasks on the different processors and performing a simulation, a user can estimate how the multiprocessor performs with different mappings. More details about this case study are in [11].

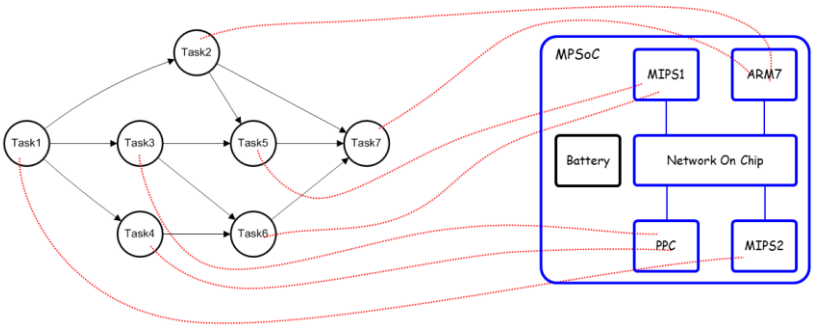


Figure 17 Mapping Tasks on a Multiprocessor Platform

7.4.2. Perform DSE with EE

Use a pre-defined POOSL model

A POOSL model is developed to specify the task graph and the multiprocessor system. Simulation of the POOSL model generates a configuration file, a Gantt Chart file, a quantity file, and a HTML file as outputs.

Define an experiment and download an EE model

Three concrete steps are involved:

- Upload the pre-defined POOSL model, which is added to the canvas named "dse".
- Define the execution steps by adding executors and connections, which are added to the canvas with blue blocks with concrete executor names.
- Click the "DOWNLOAD EE MODEL", an ".ee" file can be downloaded to the local disk.

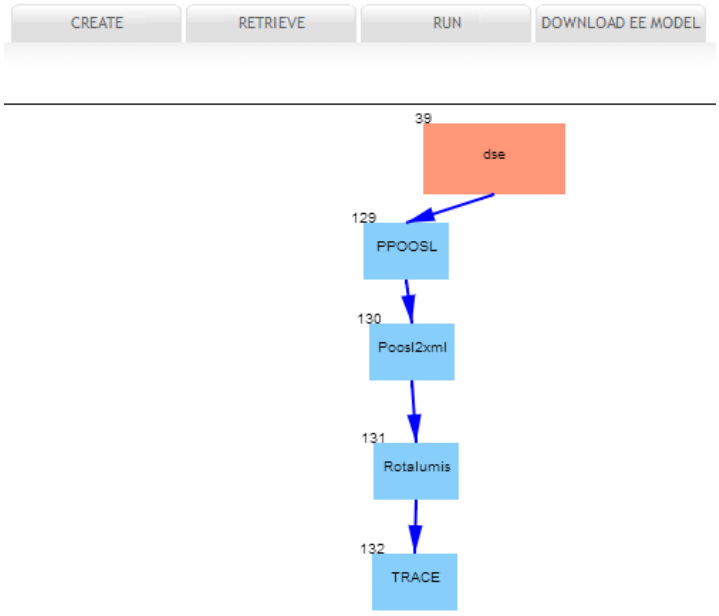


Figure 18 Define an Execution Flow with a Pre-defined POOSL Model

There are four executors in Figure 18 and their functionalities are:

- PPOOSL: setting parameter values for a POOSL model.
- Poosl2xml: generating an xml file from a POOSL model, since Rotalumis only takes an xml file as an input.
- Rotalumis: performing a simulation for a POOSL model. For this particular multiprocessor system, Rotalumis produces a set of files as inputs for TRACE.
- TRACE: visualizing the simulation results with a Gantt Chart.

Add an EE model and execute the experiment from DF

The procedure to add an EE model and execute the experiment from DF is as follows:

- Add the downloaded ".ee" file to a transformation in DF, the transformation is called "dse ee" in Figure 19.
- Create input/output parameters and connect them to the "dse ee" transformation. All the parameters listed under the block "dse" are input parameters. The output of the transformation refers to an image.
- Set all the input parameter values and execution can be triggered. In this case study, the user needs to map seven tasks to (at maximum) four nodes, specify node types, set the maximum execution time units and declare desired throughout. During the execution, the output image part displays a hour glass and the progress status appears below the image, shown in Figure 19. After the entire execution flow completes, it updates the DF UI with a Gantt Chart to display the task scheduling, see Figure 20.

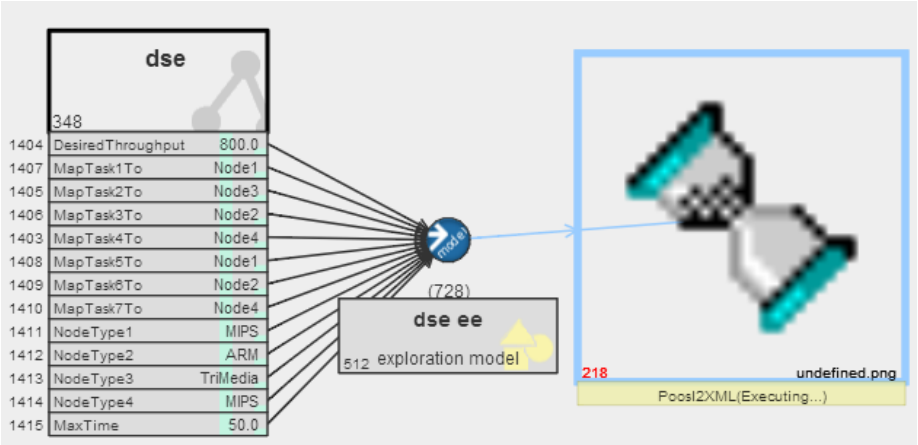


Figure 19 Running EE Experiment from DF

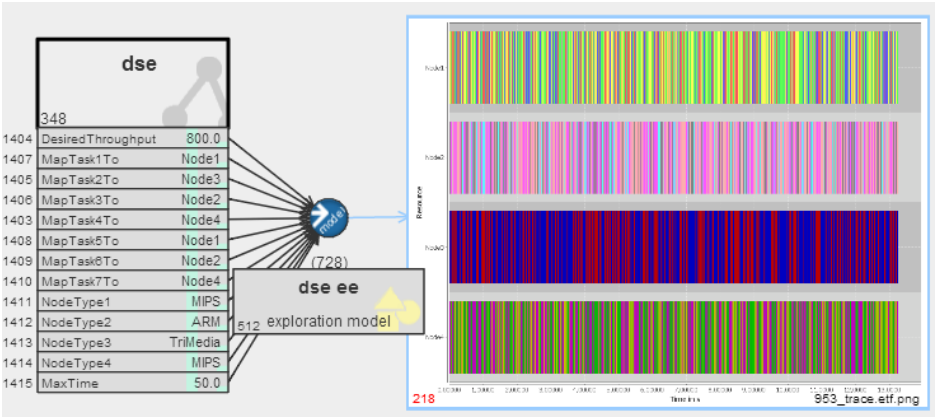


Figure 20 A Completed EE Experiment Execution from DF

By changing input parameter values, the user performs the DSE for the specific the multiprocessor system.

7.5 TRACE Validation

As mentioned in the EE Validation section, the simulation output of the POOSL model includes a configuration file, a Gantt Chart file and a quantity file, which can be visualized by TRACE. For the EE validation, only the TRACE executable file is used to generate an image, however, the Gantt Chart and quantity files can be opened in the TRACE tool separately. By changing the input values and conducting several executions, a set of simulation results are generated. These simulation results for the multiprocessor system are used to validate the TRACE tool. More details about TRACE are described in the user manual, which is available on [12].

7.5.1. Input Files for TRACE

The Gantt Chart comparison and DS graphs require at least two simulation results. In this example, five simulations are conducted. The relevant simulation output files are grouped by the run ID separately, shown in Figure 21. Either the TRACE Eclipse plug-in or the standalone application requires a root directory, thus, the "DSE" folder is created as the root directory for these five simulation folders.

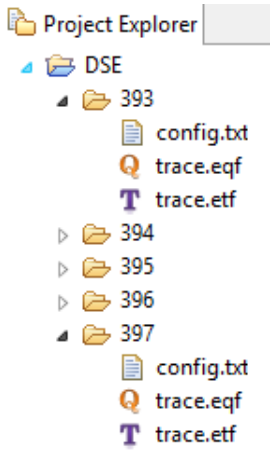


Figure 21 Simulation Output Files

For every simulation, it produces different three files:

- "config.txt": it contains the configuration information for displaying a Gantt Chart or a DS graph.
- "trace.etf": it contains the task scheduling information about how the seven tasks to be allocated to four nodes.
- "trace.eqf": it contains the statistical data like throughput, average loading of nodes etc.

7.5.2. Display a Single Gantt Chart

From the "Project Explorer", any "trace.etf" file can be visualized to a Gantt Chart by "Double Click". Figure 22 is a Gantt Chart visualization of the "trace.etf" in the folder "393". It illustrates how the seven tasks are scheduled on four nodes.

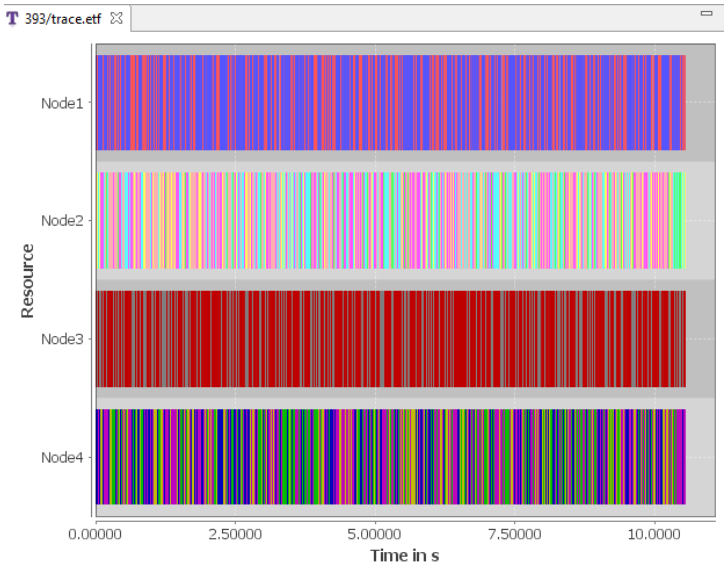


Figure 22 A Single Gantt Chart Visualization in TRACE

By interacting with the Gantt Chart, more detailed information can be displayed. For instance, zooming into Node4 with particular period and clicking on a claim, then the claim's property is displayed in the "ESI Trace Properties" view, shown in Figure 23.

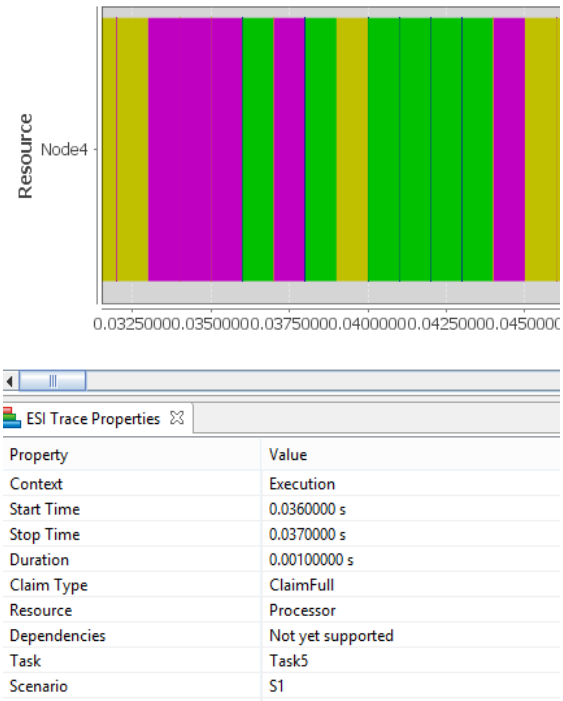


Figure 23 Display Properties of a Single Claim in a Gantt Chart

7.5.3. Open Gantt Chart Comparison

From the "Project Explorer", select multiple "trace.etf" files or folders which contain multiple "trace.etf" files. For instance, select the root directory "DSE", then the comparison includes all the five "trace.etf" files. The relevant Gantt Chart comparison is visualized in Figure 24.

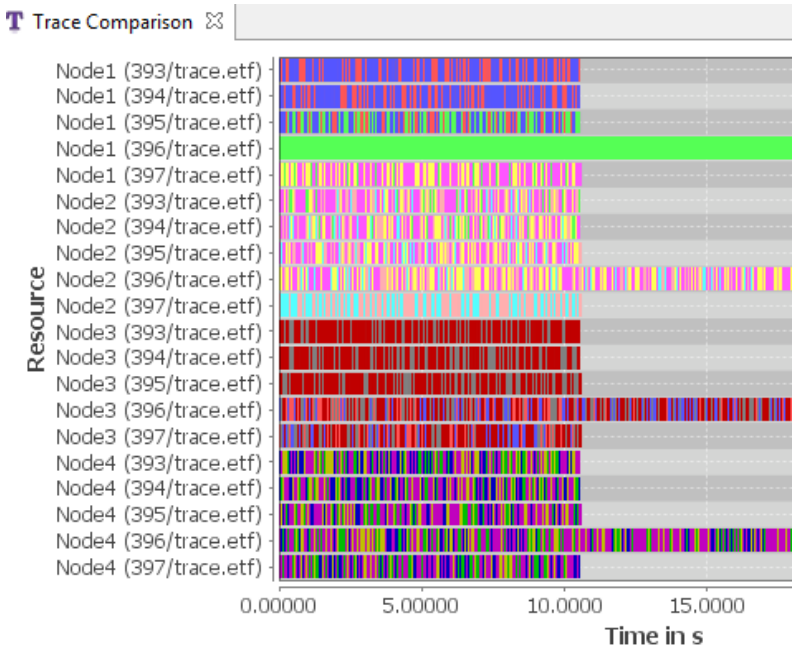


Figure 24 A Gantt Chart Comparison Visualization in TRACE

Figure 24 is an overview of the Gantt Chart comparison among five files and includes plenty of information. TRACE supports many features to check details. For example, filter the resource and only display the tasks scheduling on Node1, shown in Figure 25.

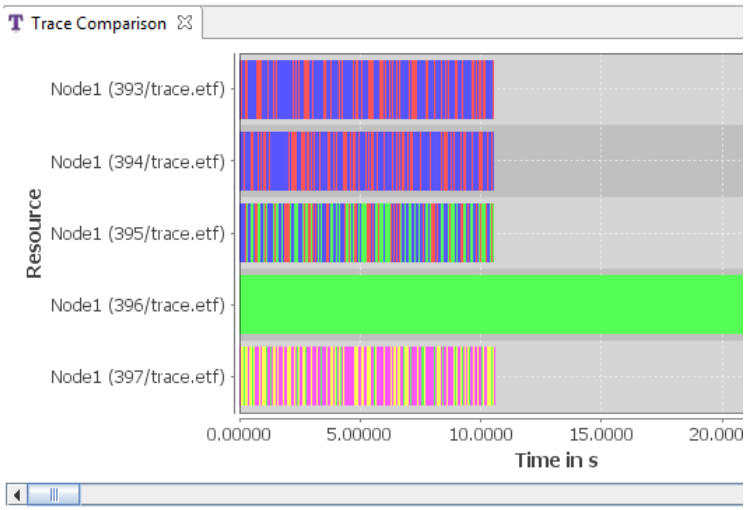


Figure 25 Display the Comparison on Node1

7.5.4. Open DS Graph

From the "Project Explorer", select multiple "trace.eqf" files or folders which contain multiple "trace.eqf" files. For instance, select the root directory "DSE", then the DS graphs contain all the five "trace.eqf" files. Select a DS graph type and quantities through a DS graph selection dialog. The selected DS graph with defined quantities can be generated. For example, select a "Radar Graph", check all of the eight quantities then a radar graph with eight coordinates are generated, see Figure 26.

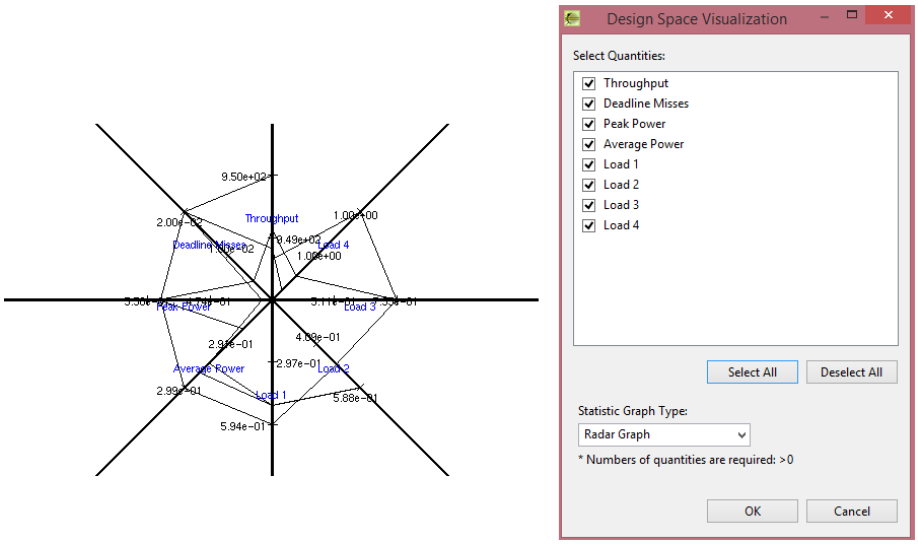


Figure 26 A DS Graph Selection Dialog and the Relevant DS Graph

With the same five "trace.eqf" files, TRACE can generate six types of DS graphs, shown in Figure 27. (1. Radar Graph, 2. 3D Scatter Plot Graph, 3. 2D Scatter Plot Graph, 4. 3D Heat Graph, 5. 2D Heat Graph, 6. Parallel Coordinates Graph)

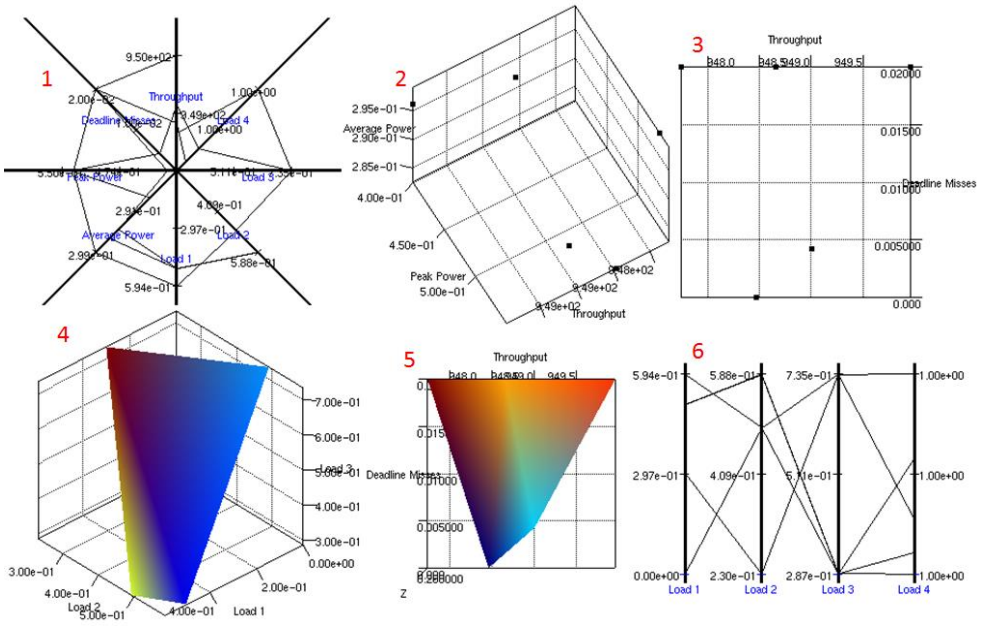
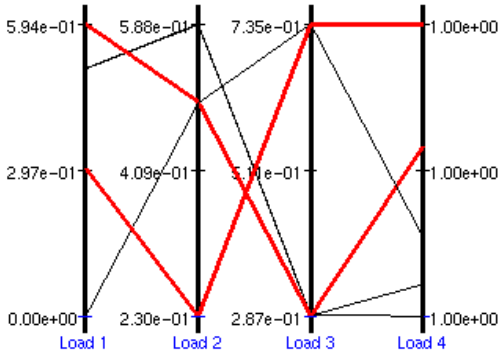


Figure 27 Six DS Graph Types

Except for heat graphs, in which statistical data from five files are merged together, points or curves in the other graphs can be selected. Once a point/curve is selected, its properties can be displayed in the "ESI Trace Properties" view. Multiple points/ curves selection is also possible by using "Ctrl" key. One example is shown in Figure 28.



ESI Trace Properties

Name	ID	Relevant Traces	Load 1	Load 2	Load 3	Load 4
395/trace.eqf	2	[trace.etf]	0.5935592	0.49286315	0.28732	1.0000103
397/trace.eqf	4	[trace.etf]	0.30092794	0.23008871	0.7343303	1.0000172

Figure 28 Display Properties of a Point/Curve in a DS Graph

Notice that the "Relevant Traces" in "ESI Trace Properties" in Figure 28 indicates which Gantt Chart file(s) the selected quantity file is (are) referred to. In this case study, a "trace.eqf" file contains statistical data for a "trace.etf" file and two files are located in the same folder with the run ID. By "Double Click" on the curve in the DS graph or "Double Click" the curve's field in the "ESI Trace Properties" view, the relevant Gantt Chart file can be open in another editor.

8.Conclusion

This chapter presents the results that have been achieved of the project and the benefits that can be gained. It also discusses the lessons learned throughout the project. Moreover, the aspects that have not been fully addressed yet are described as future work in this chapter.

8.1 Results

8.1.1. Exploration Experiment Results

Exploration Experiment(EE) as a new tool for TNO-ESI is developed within this project. At the end of the project, the concept of coherent tool support for design-space exploration was shown in a demonstration, described in Chapter 7.

The results from users' perspective can be addressed as follows:

- Define an experiment with the EE UI by specifying a flow of a model and executors.
- Download EE models and apply them to DF.
- Execute an experiment from DF by setting input parameter values.
- Execute an experiment with default parameter value through the EE UI.
- Update the DF UI by displaying the progress status during the execution and final output results after the execution.
- Update the EE UI by displaying the progress status during the execution.

The results from a developer's perspective are as follows:

- Apply the MVC architecture pattern to the EE tool.
- Separate EE Execution Handler from EE Definition Handler in order to have a stable running server to handle all the executions. In this way, the execution time for each experiment is reduced.
- Introduce a generic concept " Executor". Every executable is regarded as an executor at EE. Because of this "Executor" concept, a new executable can be added into EE easily within three steps, described in 7.2.2.

8.1.2. TRACE Results

TRACE was an existing tool from TNO-ESI, which can visualize a single Gantt Chart. The TRACE extension development brought many new features and applied a better architecture.

The results from the users' perspective are addressed:

- Build a TRACE standalone version to work apart from the Eclipse IDE.
- Support for Gantt Chart comparisons.
- Generate DS Graphs for statistical data visualization.
- Build an executable JAR file to export images.

The result from the developer's perspective is refactoring the existing TRACE code according to the MVC architecture pattern. Based on MVC, the current TRACE code is easy to extend.

8.2 Lessons Learned

The lessons I learned throughout the project came from both technical and organizational aspects.

From technical aspect, several lessons were learned:

- It is my first time to try web application development, before I only knew a little about JavaScript and HTML. Thanks to the project, I got an experience with developing a web application with PHP and jQuery .
- At the beginning of the project, reading existing TRACE and Envisioncy codes was a rough task. Besides reading relevant documents, I found the "Debug" functionalities from Eclipse IDE quite helpful. It guided me to understand the code better.
- Since MVC is applied to both EE and TRACE architectures, I have got a better understanding of the MVC architecture pattern.
- During integration with DF, some future work in DF was discovered. For example, executing an experiment should be triggered by sending a specific command instead of changing input parameter values. Terminating an running experiment from DF.

From organizational aspect, two lessons I think I can apply in the future work:

- If the requirements are not clear, communication with stakeholders and figuring out their needs are very important. If any conclusion is reached during the communication, confirm with them afterwards.
- If there are multiple tasks on hands, prioritize them and estimate the time.

8.3 Future Work

This report was based on the development of the first iteration[3] of the EE. Since it is a two person's project, my teammate continued to develop the EE tool after the first iteration. Some of the following future work has already taken into her individual work. The suggestions for future work are presented:

- Improve the EE UI. The first iteration focuses on feasibility of features, thus, a fancy UI is not a concern. However, like any other popular tool, a user-friendly interface is always preferable.

- Support for automatic exploration algorithms. The user can define a sequence of one model and executors with the current EE. From DF, user can set input parameter values to trigger the execution. In the future, EE should support a loop which can conduct several iterations automatically. By then, the user can set a range of input parameter values together from DF and EE may go through every possible input combinations automatically. At that moment, the comparison Gantt Charts and DS graphs can be exported to DF as a final result.

- Integrate with other models and executables to the EE environment. Currently, EE is applied for POOSL models, "parameterizedpoosl.jar", "poosl2xml.jar", "rotalumis.exe" and "trace.jar" executables. In the future, more models will be added, like Excel models, Matlab models, and more executables are also required when different models are added. According to the architecture, it should not be a big issue when new models are introduced theoretically, however more practical tests are required.

- Add version control to EE. The user needs to download an EE model to his/her local disk in order to apply an experiment to DF. If the same experiment has modified later on, the old EE model is not valid any more. Since there is no version control mechanism currently, the EE system cannot distinguish whether an EE model is the latest one. If every EE model has a version ID, the EE Execution Handler can check the version ID before execution.

- Extract quantities from a POOSL model. The output parameters in an EE model are from the last executor currently. In the future, the quantities from

[3] The first iteration has completed at the end of August, 2014. With this iteration, EE can define and execute a simple sequence of a model and executors with a simple UI.

a POOSL model can be extracted as output parameters in an EE model. As a result, the relevant quantity values can be displayed on the DF UI directly after executions. This future work depends on an extension of the POOSL tooling, called observers.

- Improve the TRACE tool based on user feedback. Since the TRACE tool has already released to the public since last April, some feedbacks have already collected.
    - Once a DS graph is generated, one core from CPU is always occupied until the TRACE is closed. This problem is from the OpenGL library.
    - Another common issue is from the TRACE standalone application, the "Project Explorer" is not activated at the beginning, it requires some user interactions to activate.
    - Error notifications in TRACE are not helpful, they cannot pinpoint the real cause of an error.

9.Project Management

This chapter provides some insights in various aspects related to the project management. It includes an overview of the life circle of the project, a concrete description of the work-breakdown structure and a risk management analysis..

9.1 Introduction

During this project, several methods were used to support a good project management. At the beginning of the project, it was clear that the project was divided into two parts: TRACE Extension Development and EE Development. The entire project management was also separated into two parts:

- From January 6 to April 11 - TRACE extension development
- From April 14 to September 26 - EE development

In the early stage of the project, a milestone-timeline overview was scheduled to manage the trend of the entire project. Since the management group only had some initial opinions about this project, requirements and technologies were not so clear at that moment. Therefore, an agile development method was applied throughout the entire project. In this way, it was easy to clarify the requirements gradually. A regular meeting was held every week. During the weekly meeting, the completed tasks could be confirmed, tasks in the next week could be discussed and the priority of tasks could be adjusted if necessary.

The rest of this chapter discusses the entire project planning and scheduling, work-breakdown structure and risk management.

9.2 Project Planning and Scheduling

As mentioned before, an overview of milestone timeline was made in the early stage of the project. Throughout the entire project, the timeline was adjusted several times and the final version is shown in Figure 29. It shows all the milestones and their related deadlines.

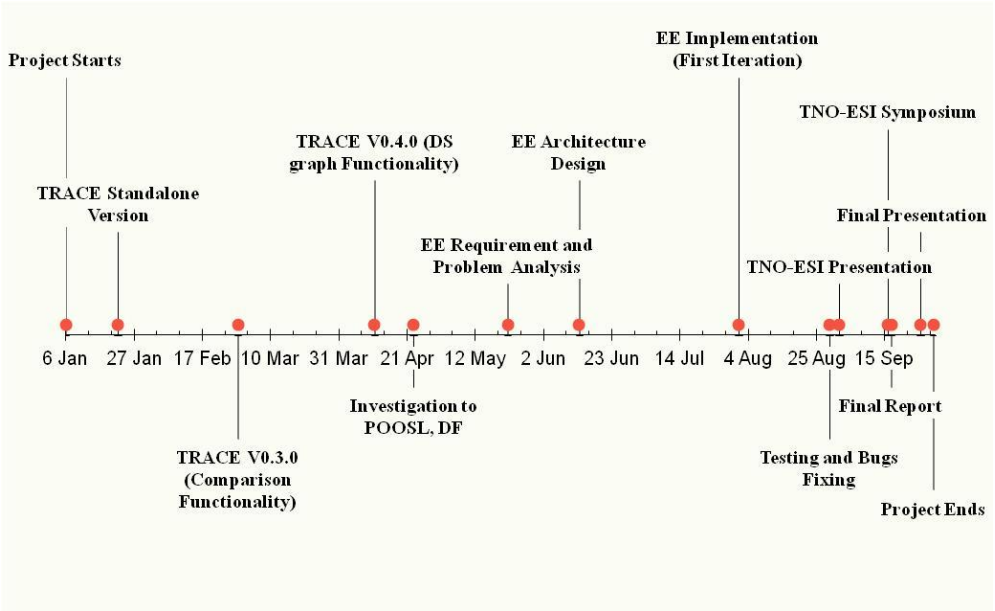


Figure 29 An Overview of Milestone Timeline

9.3 Work-Breakdown Structure (WBS)

After knowing the milestones and their deadlines, a work-breakdown structure was updated monthly. The entire project lasts for nine months, which was 38 weeks in total. Besides three week holiday, there were 35 weeks left. Since the project can be divided into two parts, the 35 weeks are also divided into two parts:

- 14 weeks - TRACE Extension Development
- 21 weeks - EE Development

Figure 30 and Figure 31 are the work-breakdown structures of TRACE Extension Development and EE Development. They illustrate how the total 35 weeks spread over the small parts. The blocks with the same color are at the same decomposition level. In Figure 31, the activities with asterisk are joint activities with my teammate.

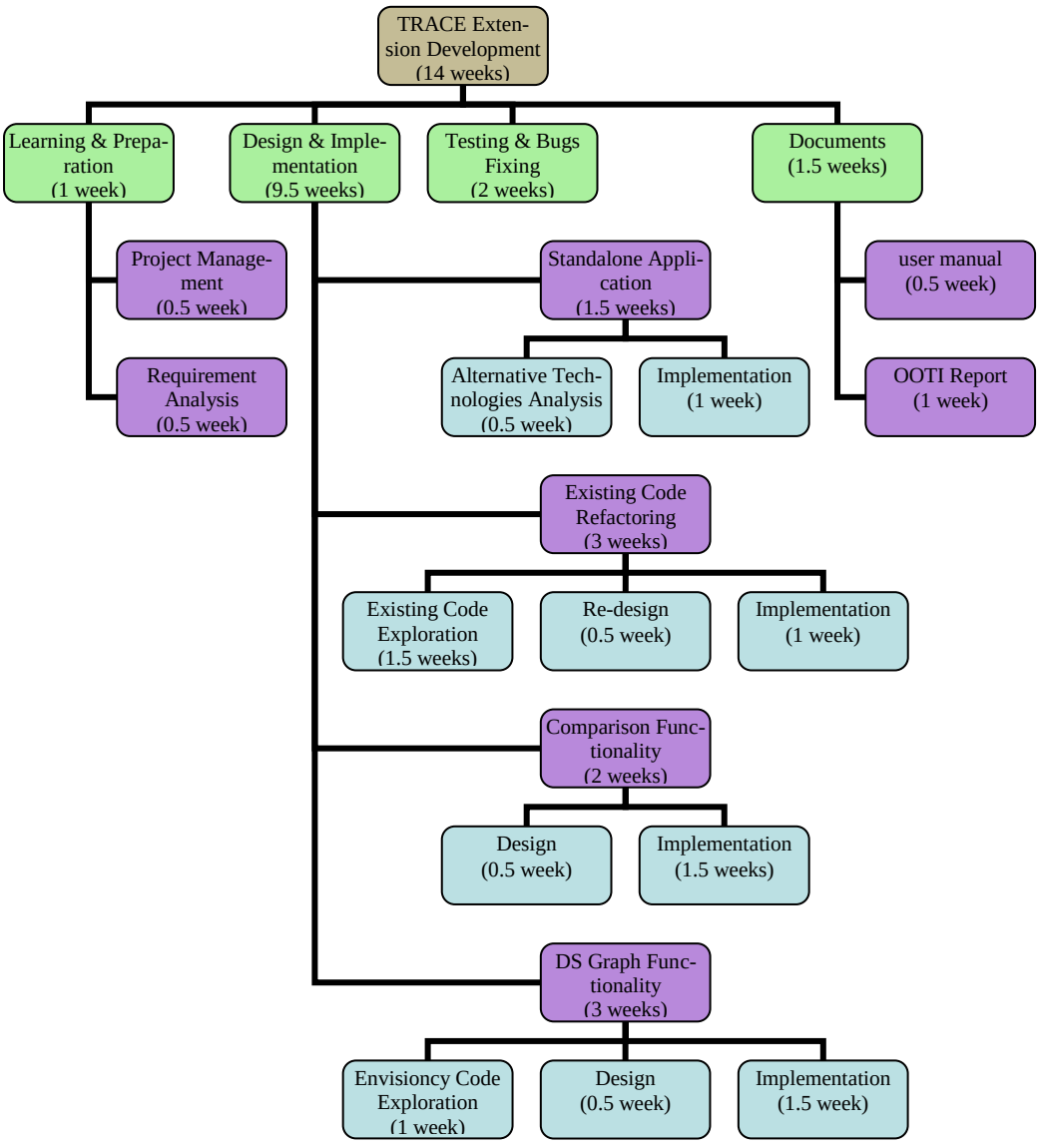


Figure 30 TRACE Extension Development Work-Breakdown Structure

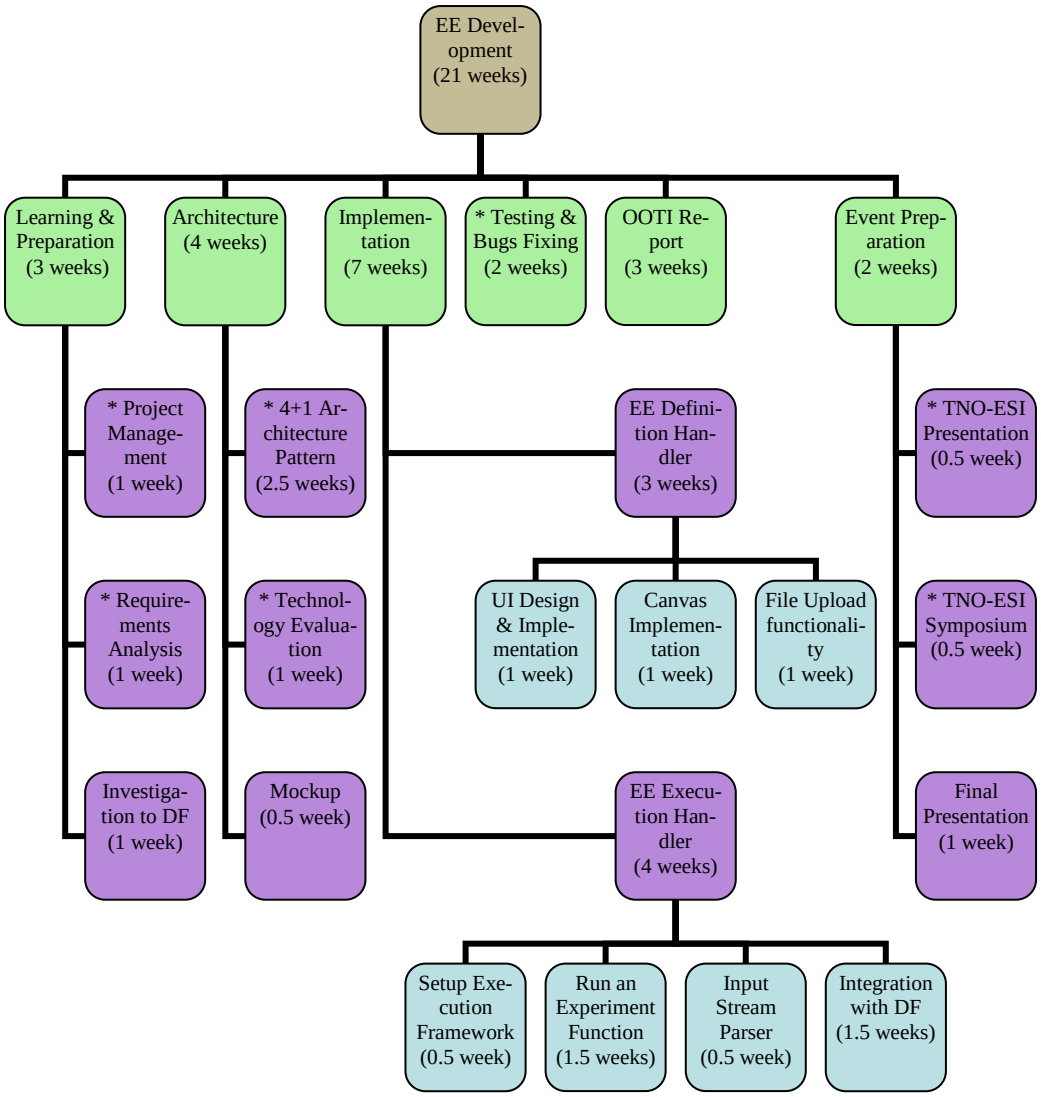


Figure 31 EE Development Work-Breakdown Structure

9.4 Risk Management

In the early stage of the project, risk analysis and the relevant avoidance/mitigation strategies were taken into account. Some risks did not appear, but other risks did occur during the project. The way to handle the risks in a proper way was to apply risk management throughout the project. The most important risks are listed in Table 22.

These risks are categorized according to their likeliness (L) and impact (I) on a scale from 1 to 5. Likeliness refers to the risk's occurrence probability; Scale 5 is the highest likelihood. Impact refers to the severity of the risk occurrence; Scale 5 is the highest severity.

Table 22 Risk Analysis and Avoidance/Mitigation Strategy

ID	Risk Description	L	I	Avoidance/Mitigation Strategy
1	EE development does not have sufficient time.	3	5	• Make sure the EE development start on time, which is planned on April 14. • Have informal discussions with stakeholders before April 14 to have some initial ideas. • Once it starts, prioritize tasks, focus on "must-have" components first.
2	Individual scope of EE	2	3	• Once the EE development starts, have dis-

	development is not clear.			cussion with my teammate early on. • Have a task-responsibility table to clarify individual tasks.
3	Requirements cannot be clarified at the beginning.	4	3	• Organize discussions with stakeholders as soon as possible. • Confirm the requirements with stakeholders after every discussion. • Show an initial prototype to clarify the requirements.
4	Required tools for the development have not been delivered in time.	3	1	• Keep an eye on the progress of the required tools. • If the new released version cannot be delivered in time, use the latest stable version instead.
5	Miscommunication with stakeholders can take place.	4	2	• Organize discussions with stakeholders frequently. • Take the notes during the discussion and confirm them with stakeholders afterwards.
6	New requirements can be added at the end of the project.	5	3	• Prioritize all the remaining tasks to check the feasibility of the new requirements. • If it is possible to take the new tasks, perform them in a good order. • If it is not possible for me to fulfill, delegate new requirements to my teammate.
7	Demonstration may fail during the TNO-ESI presentation, symposium or the final presentation.	1	2	• Practice the same use cases several times before demonstration. • Check hardware facility in demonstration rooms. • Have a working version in a backup laptop. • Record a successful demonstration in advance, in the worst case, play a video instead of a live demonstration.
8	Document support for exploration of the current version of the tool is not sufficient.	5	2	• Discuss with tool developer directly. • Use debug feature in Eclipse while exploring Java code.
9	Technical knowledge about Eclipse plug-in development is not sufficient.	2	2	• Invest some time to learn more Eclipse plug-in knowledge from websites or books as a preparation. • Ask experts or colleagues for help.
10	Technical knowledge about web development is not sufficient.	4	2	• Invest some time to learn more web development knowledge from websites or books as a preparation. • Ask experts or colleagues for help. • Divide the tasks according to the familiarity of technologies.
11	The theoretical architecture cannot be applied into practical implementation.	2	4	• Ask supervisors' feedback frequently for architecture design. • Perform sufficient experiments before finalizing the architecture. • Start from an initial architecture and improvise it during the implementation.
12	Integration with DF may not work.	4	5	• Discuss with DF team frequently and come up a integration plan together. • Clarify the interfaces between DF and EE. • Test the implemented interfaces before integration.

Figure 32 represents a classification of the risks that are shown in Table 22. According to their likeliness and impact, some risks are classified as being more critical. Risks in red circle are most dangerous, as they score high on both likeliness and impact. During the project, the most crucial risks were always paid more attention, such as Risk 12. The less serious risks could raise their scales of likeliness and impact for a certain period, therefore, the risk management was adapted accordingly. Throughout the nine months, Risk 3, 6, 8 and 10 occurred for a short period separately, but all of them were eliminated by applying the relevant mitigation strategies.

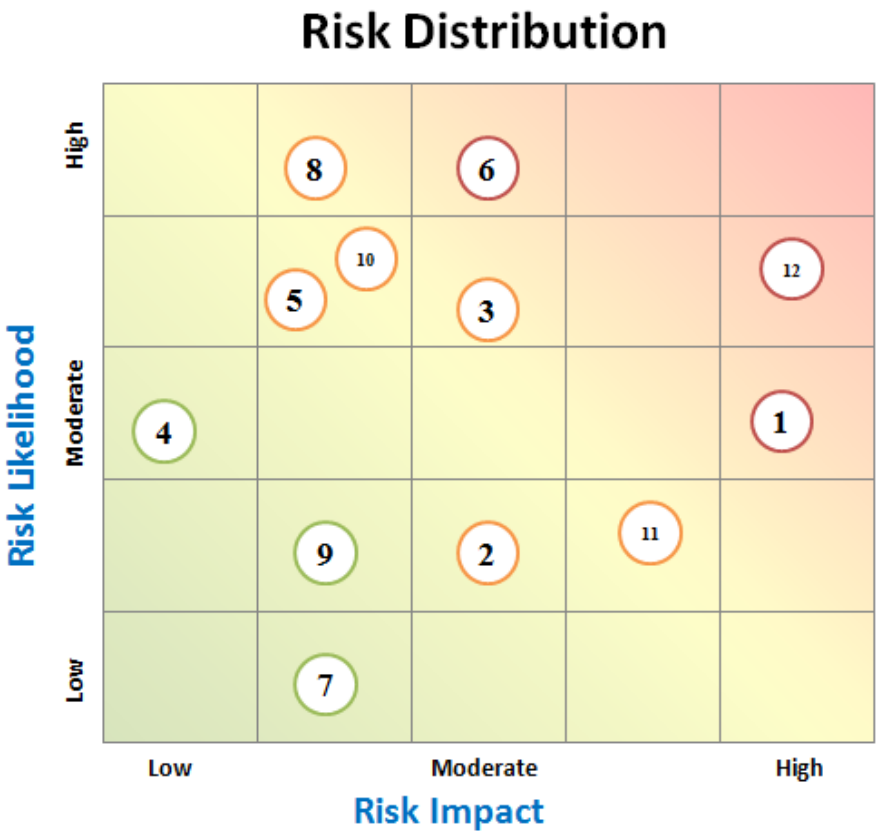


Figure 32 Risk Distribution on Likeliness and Impact

10. Project Retrospective

After providing more information about the project management process that was applied in this project, this chapter takes a look back at design criteria which were focused during the system design and implementation. Moreover, the general reflection which I realized during the project is also addressed in this chapter.

10.1 Design Criteria Revisited

A number of design criteria is selected to design the EE and TRACE tool. They are brought up in Chapter 4, and evaluated in Chapter 7. It is a right moment to revisit the design criteria and check whether they have been achieved properly.

Usability

As usability is a design criterion for both EE and TRACE, I put lots of effort to achieve it. I communicated with many colleagues at TNO-ESI to get their feedback about the user interfaces. Many exception handlers are provided to assist users when making mistakes. For EE, a simple UI to display all the necessary information is the main concern. With the simple UI, EE can fulfill all the functionalities. For TRACE, I tried to provide the same user experience for the users who were familiar with the existing features. I believe the usability criterion has been realized for both EE and TRACE parts.

Extensibility

Extensibility is also a design criterion for both EE and TRACE, since both tools are still under development and many new features need to be added. For EE, the generic concept of "Executor" makes the EE framework easy to be extended. For TRACE, I put much effort to refactor the existing codes according to the MVC architecture pattern so as to fulfill the extensibility criterion. Therefore, I think the extensibility has been achieved. However, both tools need to conduct more practical tests in the future.

Loose-coupling

Loose-coupling at EE development is the first design criterion I got at the beginning of the project. During the entire project, I always kept it in my mind. In order to achieve the criterion, I discussed with the DF team frequently to clarify how DF and EE interact with each other. On one hand, we applied clean interfaces between DF and EE to ensure they can work together properly without interfering the existing DF features. On the other hand, EE has its own web application to define and execute an experiment. As a result, DF and EE are independent with each other. This design criterion has also been approved by the DF team. I think the loose-coupling is most committed the design criteria in this project.

Generality

Generality is a design criterion for the EE development, which is a pre-condition for EE extensibility. Every executable inside the EE framework is regarded as an "Executor" without distinguishing what function the executable provides. The EE framework only concerns about which folders are input folders of an executable and which folder is its output folder. The only specialization part varies from one executable from another executable is the way to handle the inputs and the way to execute it. Since I only kept the necessary specialization part for an executable, I believe the generality design criterion has fulfilled.

Performance

Performance is a design criterion for TRACE from the user's perspective. TRACE has already been used in industry, the input files for TRACE are usually very large. The old TRACE tool usually took quite a lot of time to open them. I found the old TRACE tool loaded configuration file every time before opening a Gantt Chart file. After I noticed that, I modified the TRACE tool to check whether the configuration file has already been loaded first. As a result, for the first time to open a Gantt Chart file, it requires the same time amount, however, opening the second file is much faster. I think I solved one performance issue for TRACE and this design criterion can still be improved from other aspects in the future.

10.2 Reflection

All in all, I think this nine-month project was very successful. A case study of coherent tool support for design-space exploration was demonstrated successfully. A new version of TRACE with Gantt Chart comparison and DS graphs was released in April. Different stakeholders were satisfied with my fruitful results. Feedbacks from clients and colleagues about both EE and TRACE were very positive in general.

Personally, I had a wonderful time working at TNO-ESI. I gained much experience throughout the entire project. Since what I learned from the university was very theoretical, this project was a very practical one, which required a real release and a concrete demonstration. Applying the theoretical knowledge to a practical project was a challenge to me. From this project, I realized that I should not restrict to the book when I face a practical case. Furthermore, I learned quite a lot new technologies, such as PHP, HTML5 Canvas, web socket and web service. I also had a deep understanding of MVC architecture pattern. I believe the technical knowledge I learned during the nine months can be very helpful to my future career.

The experience is not only technical knowledge, but also about communication skills in a company environment. I realized the importance of communicating with colleagues deeply. It was really helpful to discuss my problems with colleagues. They can give their opinions which might solve the problems. Even if they were not in the same domain, I could think about my problems in a more structured way while explaining them. Moreover, communicating with colleagues helps to know each other's working styles better, which is very important for later cooperation. After nine-month internship, I believe I have already been adapted to the TNO-ESI working environment and I am happy to continue my career there after my graduation.

Glossary

4+1 Architecture view model	It organizes a description of a software architecture using five concurrent views, each of which addresses a specific set of concerns.
Apache	The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT.
DF	Design Framework is a web application which aims for system architecting including architectural views, work flow support and the link of architectural reasoning to concrete modeling activities and artifacts.
DS Graph	Design-space Graphs are a set of graphs for visualization of statistical data in design-spaces.
DSE	Design-space exploration refers to model, analyze and select appropriate design alternatives in the early phases of product development.
Eclipse plug-in	A plug-in connects with a universe of other plug-ins to form a running application in Eclipse.
EE	Exploration Experiment is a tool, which provides an integrated environment for the other tools to work together.
EE Definition Handler	It is a part of the EE tool, which is a web application and used to define concrete experiments.
EE Execution Handler	It is a part of the EE tool, which focuses on the logical rules of handling all the executions and manages the database and file system.
Envisioncy	It is a Java application, which is used to display DS graphs for statistical data.
Experiment	An experiment contains a sequence of a model and executors.
Gantt Chart	A Gantt chart is a type of bar chart, which illustrates a task schedule.
HTML5	HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web.
IDE	An integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development.

Java Archive	JAR (Java Archive) is a package file to distribute application software or libraries on the Java platform. It can be executed from the command line.
Jetty	The Jetty Web Server provides an HTTP server and Servlet container capable of serving static and dynamic content either from a standalone or embedded instantiations.
JSP	JavaServer Pages (JSP) is a server-side scripting language, based on Java.
MVC	Model-View-Controller design pattern.
MySQL	It is a widely used open-source relational database management system.
OpenGL	Open Graphics Library is a cross-language, multi-platform application programming interface (API) for rendering 2D and 3D vector graphics.
PHP	PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language.
POOSL	It provides an integrated editing, debugging and validating environment for POOSL modelling, combined with a high-speed simulator.
Rotalumis	It is the a high-speed simulator for POOSL models.
TRACE	It is a tool for visualizing quantitative analysis results.
URL	A uniform resource is a specific character string that constitutes a reference to a resource. Most web browsers display the URL of a web page above the page in an address bar.
WampServer	It is a Windows web development environment, which allows developers to create web applications with Apache2, PHP and a MySQL database.

Bibliography

[1] TNO-ESI “About” webpage. Available: http://www.esi.nl/about-tno-esi/.

[2] TNO-ESI website. Available: http://www.esi.nl/solutions/.

[3] TNO-ESI DSE webpage. Available : http://dse.esi.nl/.

[4] DF webpage. Avaiable: http://df.esi.nl/.

[5] TNO-ESI POOSL webpage. Avaiable: http://poosl.esi.nl/.

[6] TU/e POOSL webpage. Available : http://www.es.ele.tue.nl/poosl/.

[7] TNO-ESI TRACE webpage. Avaiable: http://trace.esi.nl/.

[8] WampServer webpage. Avaiable: http://www.wampserver.com/en/

[9] PHP-MVC basic version github website.
Available: https://github.com/panique/php-mvc

[10] FTP wiki webpage.
Available: http://en.wikipedia.org/wiki/File_Transfer_Protocol

[11] Assignment 2 - Deign-Space Exploration. Available:
http://www.es.ele.tue.nl/~btheelen/education/5kk80_assignment2.pdf

[12] TRACE user manual webpage.
Available: http://trace.esi.nl/documentation.php

[13] EclipsePluginSite.com, “Eclipse Plugin Development” website.
Available: http://www.eclipsepluginsite.com/index.html

[14] H. Moneva, R. Hamberg and T. Punter, “A Design Framework for Model-based Development of Complex Systems”, 32nd IEEE Real-Time Systems Symposium, 2nd Analytical Virtual integration of Cyber-Physical Systems Workshop, Vienna, 2011.

About the Authors

Fangyi Shi received her MSc diploma in Software Engineering from Technical University of Denmark in 2011. Her Master thesis was titled “Translating Communicating Sequential Processes to Formal System Design Models”, which researched a mapping strategy from a high level language to a system design model. She obtained her BSc diploma in Information Security from Beijing University of Technology in 2009. Her Bachelor thesis was about face recognition based on artificial neural net. During her Bachelor and Master studies, she took two internships, and had some experience at software quality assurance and web establishment. Her main research interests include real-time system development and software design.

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.