CoDex

3TU. School for Technological Design

STAN ACKERMANS INSTITUTE

CoDeX
Coherent Tool Support for Design-Space Exploration

Bayasgalan Baatar
December 2014

TU Delft
Delft University of Technology

TU/e
Technische Universiteit Eindhoven University of Technology

UNIVERSITY OF TWENTE.

CoDeX

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
Ramon Schiffelers

Date

December 2014

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
Software Technology program, MF 7.092
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31-(0)402474334

Published by
Eindhoven University of Technology
Stan Ackermans Institute

Printed by
Eindhoven University of Technology
UniversiteitsDrukkerij

ISBN
978-90-444-1339-7

Abstract

Embedded Systems Innovation by TNO (TNO-ESI) developed three generic tools that are actively being professionalized to improve industrial applicability: POOSL, Trace, and Design Framework (DF).

With the help of the three tools, a designer can design complex embedded systems by specifying systems requirements with DF, developing and debugging models with the POOSL, simulating models with the POOSL simulator and visualizing analysis results with the TRACE.

However, there is no integrated environment to support these three tools to work together and the space of possible solutions is normally very large, i.e., many design alternatives exist. As a consequence, conducting a manual exhaustive search of the design space is prohibitive and automated DSE techniques have to be used to find a "good" solutions.

The project proposes a prototype to demonstrate cooperation between the three tools to conduct automated DSE. The report describes the process of design and implementation of a new tool, Exploration Experiment. With help of the new tool, user can provide the system all design alternatives of interest at once and system conducts automated DSE with help of the three tools to help the user to find "good" design solution.

Keywords
Design-Space Exploration, Design Framework, POOSL, Trace, Data Flow, Automated Design-Space Exploration, Exploration Experiment, Coherent Tool Support

Preferred reference
Bayasgalan Baatar, CoDeX: Coherent Tool Support for Design-Space Exploration. Eindhoven University of Technology, SAI Technical Report, December, 2014. (978-90-444-1339-7)

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” (CoDeX) 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, Bayasgalan worked in close cooperation with her OOTI colleague and team mate Fangyi to design and implemented as a team 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 are especially grateful for her contributions to the architecting and tool design during this first part. In the second part, Bayasgalan realized a much improved elegant and intuitive graphical user interface to support definition and execution of experiments, which – together with a concrete example – is an excellent demonstrator to our customers.

We enjoyed the fruitful and well-prepared progress meetings and steering group meetings. Bayasgalan has successfully finalized her assignment by delivering the Exploration Experiment tool together with this final report and requested development documentation. We wish her a lot of success in her future career.

Eindhoven, November 2014

Frans Reckers, Bart Theelen
Embedded System Innovation by TNO

Preface

This report was written to provide a detailed summary of an eight-month project carried out at Embedded Systems Innovation by TNO (TNO-ESI), Eindhoven. The project serves as a final project for the Software Technology (ST) program for the Eindhoven University of Technology (TU/e), toward the Professional Doctorate in Engineering (PDEng) degree. As such, the project serves two different purposes: the author should demonstrate her capabilities in independently carrying out a design project, while providing a sufficient added value for TNO-ESI. Therefore, the report not only provides an account of design process, but also dives deep into the problem, domain, and requirement analysis and provides various design decisions with the explanation of the rationale behind them.

Date: November 11, 2014

Acknowledgements

This project could not have been completed without the help and guidance of several people.

First of all, I am heartily thankful to my company supervisors, Frans Reckers and Bart Theelen, for giving me the opportunity to carry out this project at TNO-ESI. They devoted a great deal of their time to steer this project. I could not have carried out this project without their valuable suggestions, precise directions, and kind encouragement.

I am sincerely grateful for my university supervisor, Ramon Schiffelers, for his time and effort to guide my direction and his invaluably constructive feedback and great ideas, which shaped the project much during the steering group meetings.

I am thankful for Judith Strother for reviewing my report.

I would like to thank my teammate and a friend, Fangyi Shi, for her great effort and contribution. It was my great pleasure to work with her in this project.

I am thankful for TNO-ESI colleagues who made my time at TNO-ESI very enjoyable.

I am also grateful for Ad Aerts for giving me the opportunity to be a part of the OOTI program. Special thanks for Maggy de Wert for her support and help in the good and bad times during the program.

I would like to thank my all OOTI colleagues with whom I spent wonderful time together during the last two years. I learned a lot from them.

I am grateful for Yanja Dajsuren and Razvan Dinu for their great advice, guidance, support, and help during last two years.

Also, I must acknowledge my entire family for their support, love, and encouragement. Specially, I am heartily grateful to our mothers, Altanzaya Gunsen and Tuya Yondon, and my cousin Haliun Mainbayar for their time and dedication to support my study.

Finally, I would like to acknowledge my husband and a best friend, Tamir and the only person who can cheer me up in any condition, my son Orgil. I would not have finished this project without their love and encouragement.

Date: November 15, 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.

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 when designing these kinds of complex embedded systems.

Contributions from industry-as-laboratory projects, TNO-ESI developed three generic tools that are actively being professionalized to improve industrial applicability: POOSL, Trace, and Design Framework (DF).

With the help of the three tools, a designer 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 the CoDeX project is to demonstrate cooperation between the three generic tools as an integrated environment for DSE.

In addition to the current tools, the new tools are going to be introduced in the future to support DSE. Therefore, the environment should be able to be easily extendable with any kind of tools. One example of an extension achieved in the scope of this project is a simple algorithm for automated DSE. User can provide the system all design alternatives of interest at once and the simple algorithm automates the entire experimental process of finding most optimum one from that set.

In order to meet the goal, a new tool, Exploration Experiment (EE), was designed and developed. The first version of the working prototype was achieved together with my teammate Fangyi Shi. Then, the prototype was improved with a user friendly GUI and extended with the algorithm for automated DSE.

Besides the main functional requirements, a few non-functional requirements shaped the overall architecture and design. The system should be *modular* to be loosely coupled with other tools. The system should be *modifiable* to adapt to the future possible extensions. The system has to be *available* during the entire execution, no matter how big the execution is. The system has to be *user friendly* and elegant.

The solutions for modularity and modifiability focus on the Dataflow architectural pattern and clear interface definitions between the components. The solution for availability focuses on the Client-Server architectural pattern. The solution for usability focuses on the Event-Driven architectural pattern and a user friendly GUI implemented with a graph drawing library, jsPlumb.

Also, the design and implementation of a loop back functionality, which makes the system able to conduct an automated DSE with help of the algorithm, is detailed in this project.

As a result of the project, the automated DSE with a simple algorithm is demonstrated as a proof of the concept of coherent tool support for design-space exploration.

Table of Contents

List of Figures

List of Tables

1.Introduction

Abstract – In this chapter, a short introduction of this project including the relevant background information, the objective and scope of the project is given. Also, an outline of the entire report is provided in this chapter.

1.1 Context

Today, embedded systems impact almost every aspect of our lives: how we work, live, and communicate. Embedded systems designers always aim for higher performance at a low cost. However, it is not easy to accomplish. Mostly, there are a tremendous number of possible design choices and alternatives and it is extremely challenging to make a good decision within that huge design space. Therefore, a Coherent Tool Support, the main focus of this report, is needed to ease that process and help the embedded system designers to achieve good design solutions.

1.1.1. 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 both national and international academia.
TNO-ESI contributes to society and the economy by driving advances in high-tech system technology, with a strong shared research program, dedicated innovation support, a focused competence development program, and various knowledge and experience sharing activities [1].

1.1.2. TNO-ESI Tools

TNO-ESI is developing generic model-based design tools for embedded systems. The following are the three generic tools that have been developed by TNO-ESI with collaboration with other companies [2].

- ***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 for concrete modeling activities and artifacts.

1.2 The Project

1.2.1. The Starting Point

Current versions of the TNO-ESI tools for Design Space Exploration (DSE) are already being applied at industrial partners of TNO-ESI, whilst still being under development. Also, new DSE-purpose tools are going to be introduced in the future. For example, algorithms are going to be added to support the automated DSE process. Therefore, a framework environment is needed that can integrate any kind of DSE tools together.

1.2.2. The Objective

The goal of this assignment is to design and develop a framework environment that controls the entire flow of the DSE process by integrating and executing various kinds of DSE tools. The tools can be implemented in any kind of technologies; however, the integrated environment should not be dependent on the specific technology or platform. A concrete demonstrator for automated DSE is needed as a proof of concept.

1.2.3. The Scope

Even though the framework environment should support any kind of tools, the concrete demonstration has to focus on the main TNO-ESI tools (POOSL, TRACE, and DF.)
Each of the three TNO-ESI tools provides an essential cornerstone in a DSE and must remain independent since they can also support embedded system design individually.
The extension of these tools is expected to support integration.

1.3 Outline

This report is organized as follows:

Chapter 2 provides a domain analysis which gives the explanation of DSE and automated DSE, TNO-ESI tools and their relevance with DSE, and an introduction of DSE example of multi-processor system. This example was used to demonstrate the automated DSE and validate the architecture and design achieved in this project.

Chapter 3 describes problem analysis identifying the main stakeholders and their concerns, the problem statement in deeper level, and design opportunities.

Chapter 4 gives feasibility analysis that discussed about issues and challenges, and risks with the mitigation strategy in the project.

Chapter 5 gives a comprehensive description of the requirements in this project. High level functional requirements and non-functional requirements are listed.

Chapter 6 focuses on the overall architecture detailed by 4+1 views. The chapter also includes alternatives for different architecture patterns and their justifications.

Chapter 7 describes detailed design including number of design alternatives and their evaluations.

Chapter 8 focuses the design change and improvement of the system caused by new requirements came up during the project.

Chapter 9 verifies and validates the result against the requirements provided in Chapter 5. The case scenario described in Section 2.5 is used to verify and validate the system

Chapter 10 concludes the result of the project and proposes the future work list for the further improvement of the project.

Chapter 11 discusses project planning and scheduling, schedule change, and its impact on the project.

Chapter 12 gives the author's reflection on this project which includes the lessons learned throughout the project. Also, design opportunities described in Section 3.3 are revisited in here to check whether they have been carried out successfully.

2. Domain Analysis

Abstract – In this chapter, the introduction of basic terms that are needed to get a better understanding of the report is given. This is composed of a short introduction of a DSE, extended introduction of TNO-ESI tools and their relevance with DSE, and an example that explains the DSE for multi-processor application.

2.1 Design Space Exploration (DSE)

Design Space Exploration is an important factor in embedded systems design. During several steps in a design flow, designers have to decide between design alternatives. The decisions are located at various levels of abstraction. In addition, the decisions affect several design goals; therefore, represent a multi-criteria decision problem.

To judge the quality of a new design, the performance of a system for a given application is one core criterion. The design spaces usually involve multiple metrics of interest (timing, resource usage, energy usage, cost, etc.) and multiple design parameters (e.g., the number and type of processing cores, sizes and organization of memories, interconnect, scheduling and arbitration policies). The relation between design choices on the one hand and metrics of interest on the other hand is often very difficult to establish, due to aspects such as concurrency, dynamic application behavior, and resource sharing. No single modeling approach or analysis tool is fit to cope with all the challenges of modern embedded-system design [3].

A model-driven DSE introduced by TNO-ESI with their project named Octopus [3] greatly contributes solving the above challenge. The concept of a model-driven DSE is illustrated in Figure 1. Designers start with concepts and requirement analysis. This is done in a stepwise process where in each, the designs for particular selection of the requirements are explored. To do this, the designer needs to develop and debug models of design alternatives. After several iterations of validating the requirement and modifying models, well-defined models are developed. Then property evaluation is conducted to get exploration results. After analyzing and interpreting exploration results, the developer may reach the final design decisions or may become aware of some issues in the concept and requirement level and then improve them.

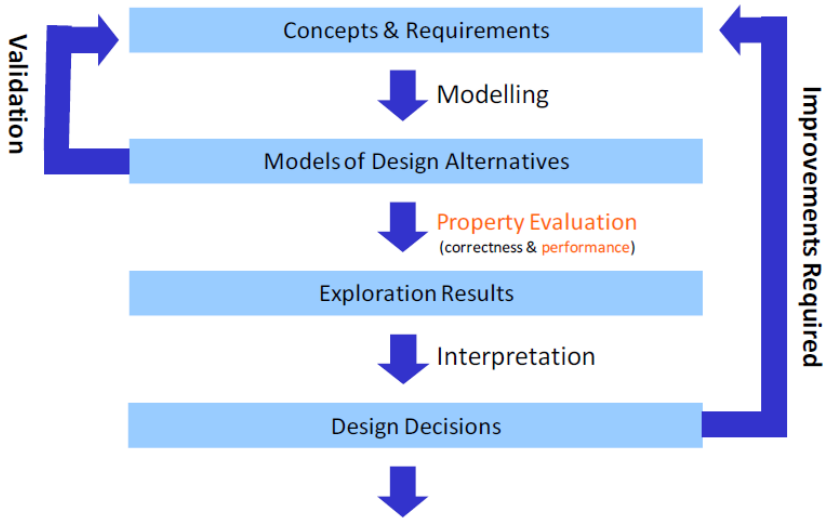

Figure 1 Concept of Model-Driven Design Space Exploration

2.2 An automated DSE

In DSE, the space of possible solutions is normally very large, i.e., many design alternatives exist. As a consequence, conducting a manual exhaustive search of the design space is prohibitive, and automated DSE techniques have to be used to find "good" solutions.

As an automated DSE technique, algorithms can be used. User can provide all range of design alternative of interest at once and the algorithm can evaluates each design alternative against the design criteria. Then, provide the user the best design alternative based on the evaluation.

2.3 TNO-ESI Tools

2.3.1. 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 encompass design decisions and analysis the results, by linking design goals to concrete questions and analysis results for a particular scope of the system. The DF also provides 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. [4]

2.3.1.1 DF concepts

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 [5]. 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.

2.3.1.2 Simple DF project example

Figure 2 shows a simple DF project "demonstrator" with tree style decomposition. Parameters “M1” and “M2” from two blocks “memory 1” and “memory 2” are connected as inputs to the transformation with 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” (_m2) is larger than “M1” (_m1) is validated. By modifying the values of “M1” or “M2,” the values of “latency” and “throughput” is updated after the execution of the transformation has been completed.

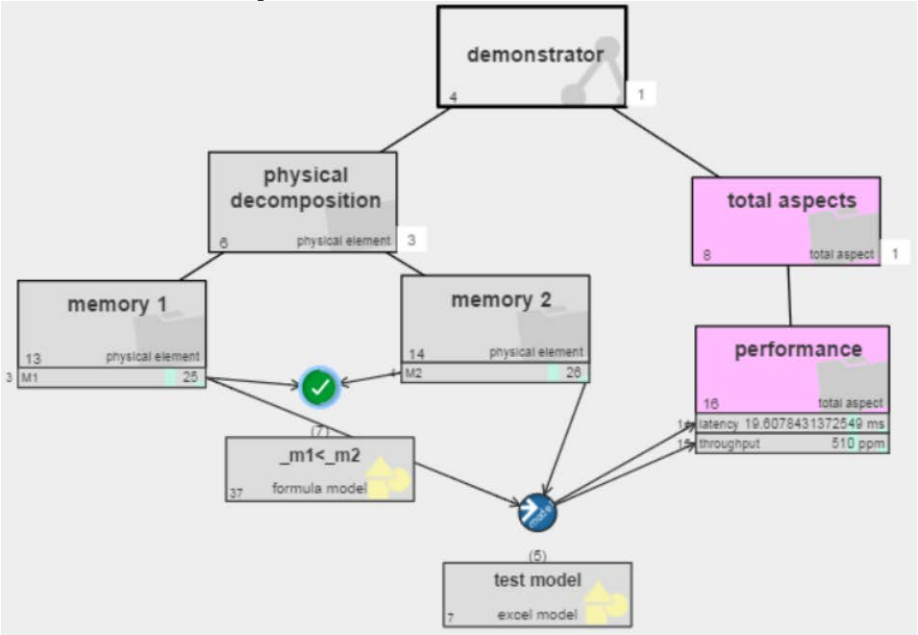

Figure 2 A simple DF project

2.3.2. 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 details about POOSL are described in [6] and [7].

2.3.3. Trace

TRACE is a design space 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.

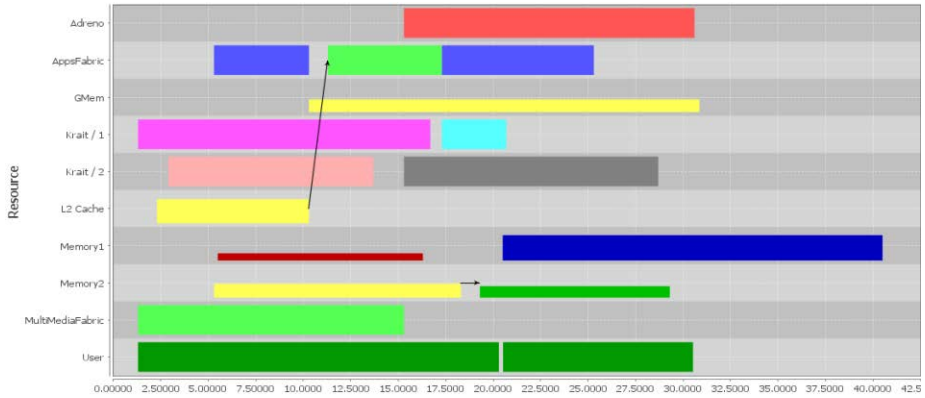

Figure 3 A simple Trace visualization output

In Figure 3, the horizontal line denotes time and vertical lines denotes resources. The colored boxes denote tasks executed on the resources within a particular timeslot. The arrow denotes a task dependency between the source task and the target task meaning that the source task has to be finished before the target task can start processing.
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 [8].

2.4 TNO-ESI Tools in DSE

Each of the TNO-ESI tools has an important role in the model-driven DSE process described in Section 2.1.
The tools can refer to specific 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** – tracks complete flow from "Concepts & Requirements" to "Design Decisions."

Currently, each step of this process is done separately and manually. The goal of this project is to design and develop an integrated framework environment that controls the flow of the entire process by integrating the above three tools.

“Improvements Required” part is done manually by user. However, with help of the automated DSE, this process can be done automatically by the system. Therefore, the integrated framework should support automated DSE and demonstrate it with a simple algorithm.

2.5 An example: DSE of a multi-processor system

In this section, the introduction of an example application, the DSE of a multi-processor system, is given to explain the model-driven DSE in a real situation [9]. Also, a similar example can be found on [10].

2.5.1. Introduction

A multiprocessor system consists of parallelized *applications* that are *mapped* onto the multiprocessor *platform*. The overall performance of the system depends on these three aspects; application, mapping, and platform. The mapping determines, for example, which processor unit of the platform executes what tasks of the application. After developing a model incorporating all three parts, the obtained performance results may give hints to improving the application, platform, and/or mapping. Iteratively applying such improvements leads to finding an optimal design solution.

2.5.2. Tasks

Figure 4 illustrates application and platform dependency of an example application used in this project. There are seven tasks and each task node represents a certain functionality (like decoding or filtering) that can potentially be executed in parallel with other tasks. The edges represent dependencies between the tasks.

2.5.3. Platform

The platform considered in this assignment concerns a Network-on-Chip (NoC) based Multi-Processor System-on-Chip (MPSoC) in a battery-powered embedded device. Four kinds of resources can be distinguished: *processor units*, *communication units*, *storage units* and an *energy source*.

2.5.4. Mapping tasks to platform

The platform has several parameters that can be set to obtain an optimal realization of the multiprocessor system. In addition to the number of processor nodes, there is a choice of three different processor types: MIPS, ARM7 and TriMedia.
The mapping describes which tasks of the application are mapped onto which processor nodes (Figure 4).

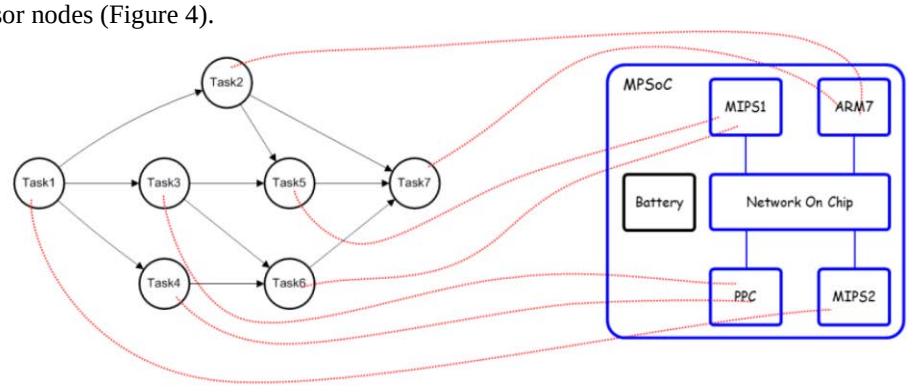

Figure 4 Task Mappings on Multi-processor Platform

2.5.5. Performance constraints

The mapping that satisfies the following performance constraints is considered as an optimum design choice.

- Throughput: *higher than* 800 (The throughput indicates the average number of firings of Task7 per second)
- Missed deadline probability: lower than 5%

3. Problem Analysis

Abstract –In this chapter, the detailed analysis of the actual problem is described. First, a stakeholder analysis is described. Then, various problem statements have to be solved in the scope of this project are given. Finally, the design opportunities needed to solve those problems is described.

3.1 Stakeholder Analysis

The following are the description of the main stakeholders in this project.

3.1.1. TU/e

The Eindhoven University of Technology is responsible for the educational aspect of the project. The university supervisor is the main stakeholder in this category who should make sure that the design and documentation meet the standards of a PDEng project.

3.1.2. TNO-ESI

TNO-ESI, as the owner of the project, wants to have good architecture and design solutions for CoDeX and a concrete demonstrator as a proof of concept. With the demonstrator, they can further analyze the customer needs at a deeper level, moreover, attract the potential customers. Also, they want sufficient documentation, such as user manuals and development documents.

3.1.3. Potential End Users

Currently, TNO-ESI's tools for DSE are already being used individually by their industrial partners. Those customers are potential end users of the coherent tool support for design-space exploration and their concern would be a user-friendly interaction among these tools.

3.1.4. Tool 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 necessary to ensure the fewest integration points.

3.1.5. Teammate

Fanyi Shi is an OOTI trainee who also worked on this project during the last five and half months of her final project. Since this is a two-person project, we need to divide our tasks clearly and collaborate with each other.

3.2 Problem Statement

3.2.1. POOSL transformation in DF

Currently, DF can work with an Excel, Matlab, and Formula transformation. The user uploads an Excel model as a transformation to conduct an experiment. In addition to that, the POOSL model has to be used as a transformation in DF. Introducing a POOSL transformation should not affect the current architecture of DF.

3.2.2. Parameters and values

A parameter and its values are important ingredients in this project. According to the example described in Section 2.5, the application tasks have to be allocated on certain platforms to evaluate the performance.
In DF, the above process is done by assigning a value to parameters. The user can do various kinds of assignments including task allocation on the platform, platform type definitions, and task prioritizations.

Since, the POOSL model is used in this project the mechanism to assign parameter values from DF to POOSL parameters is needed when a user wants to do an experiment.
For that, followings steps have to be done:

- User: upload his/her POOSL model to the system
- System: extract parameter names from the model and send them to DF
- User: set parameter values
- DF: send parameter values to the system
- System: assign parameter values to POOSL parameters

To achieve this, the POOSL tool has to be extended with “Extract Parameters” and “Set Parameter Values” functionalities and those functionalities have to be linked with DF.

3.2.3. Tool variations

The following are tool variations which are going be used in the system:

- Model converters
- Simulators
- Visualizer
- Algorithms

These tools can be implemented in any kind of technologies and might require a certain platform to run. Furthermore, totally new tools might be added in the future. Therefore, the framework should provide an environment that can integrate any kinds of tools together.

3.2.4. Long execution

The tools might take a long time to execute. Usually, POOSL models can be huge and the simulation process can take a few days. Therefore, a framework environment should be able to handle this long execution.

3.2.5. Trace result on DF

After the experiment is finished executing, the result should be shown on the DF. Currently, DF shows string values as a result of the Excel experiment. In addition to that, picture files are produced by TRACE should be shown on the DF.

3.2.6. Loops

The integrated environment should support loop back connection to achieve automated DSE where the algorithms need to iterate in the flow

.

3.2.7. Intuitive experiment definition

The user has to define a flow of the experiment by connecting all these tools together. An intuitive way of drawing this workflow is needed to provide an ease of use.

3.3 Design Opportunities

Based on the problem statement described in the previous section, the assessment criteria for the technological design that is most appropriate for this project can be identified. According to the *"New criteria for assessing a technological design"* by Kees van Hee and Kees van Overveld [11], the following design opportunities were picked out as the important ones for this project. These criteria are revisited and reflected on at the end of this report.

Important design criteria for this project:

- Functionality: What does the artifact do for its environment?
    - Satisfaction: Designed artifact has to satisfy the requirements.
    - Ease of use: the final product has to be easy to use, install, and maintain. Therefore, user friendly interface design, sufficient development documentation, and a user manual are needed.
- Construction: How will the artifact do this?
    - Structuring: The architecture has to be modifiable which can adapt to future possible extensions. Also, the components have to be modular, loosely coupled with each other.
    - Convincingness: The result of the project has to prove that the idea of "Coherent tool support for DSE" works. In other words, the prototype has to show that the TNO-ESI tools described in Section 2.3 can work together.
- Realizability: How can the artifact be realized?
    - Technical realizability: The real working prototype is needed to prove the designed architecture is technically possible.
- Presentation: What does the artifact look like?
    - Elegance: The product should look elegant and appealing to attract customers.

However, rest of the design criteria listed below are less important in this project:

- Functionality: What does the artifact do for its environment?
    - Reusability
- Construction: How will the artifact do this?
    - Inventivity
- Realizability: How the artifact can be realized?
    - Economical realizability
- Impact: What are the risks and benefits of the artifact for its environment?
    - Social impact
    - Risks
- Presentation: What does the artifact look like?
    - Completeness
    - Correctness

4. Feasibility Analysis

Abstract – In this chapter, the main issues and challenges expected during the project are presented. A list of risks is also detailed, along with the mitigation strategies.

4.1 Issues and Challenges

Following is the list of challenges in this project.

4.1.1. Architecture for cross platform/language application

Structure of the components and technology choices are important to achieve a generic architecture for cross platform/language application. Possible technology choices for investigation are: Thrift, Websocket, and Webservice.

4.1.2. Unknown future possible extension

The framework has to be modifiable to adapt future possible extensions that are not known yet. Therefore, it is important to validate the architecture with example extensions.

4.1.3. Communicate with DF

The system has to interact with a web application, DF. 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.

4.1.4. Deploy and run all the executables

Various 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.

4.1.5. New technologies

The context and problem analysis already hint that the solution leverages new technologies, or at least, a framework with which the author has no experience. Becoming familiar with them as well acquiring the broad understanding that this project requires is a challenge. Significant effort is needed to make up for those shortcomings.

4.1.6. Complex data structure

In this project, a database contains important application logics. Good knowledge of the database is needed to achieve a better design.

4.1.7. Poor documentation of existing tools

Existing tools that are used in this project are still under development; therefore, documentation is not sufficient. Therefore, active interaction with tool developers during their busy schedule is a challenge.

4.1.8. Unfamiliar domain

The author was not familiar with the domain, DSE and TNO-ESI tools, of this project. Having sufficient domain knowledge is important to analyze the requirement correctly. Therefore, it is important to learn the domain quickly by communicating with the stakeholders frequently and to read from the relevant sources.

4.2 Risks

Following is a short overview of the risks that were identified early. Also, mitigation strategies to limit their impact on the project are described.

4.2.1. Not clear/not efficient individual scope division

Scope division is one of the challenges in this project and a wrong or inefficient scope division can cause a delay in the project which can lead to project failure. To avoid that, developers need to communicate frequently. Also, sharing their background knowledge and experience is needed to allocate tasks efficiently.

4.2.2. Architecture change because of the TNO-ESI tool update

TNO-ESI tools are developing and changing every day. Therefore, those changes might cause a big impact on architecture and design. To avoid that, detailed discussion and agreement on the interface with the developers is needed.

4.2.3. New requirements at the last moment

Prototyping makes more problems clear, which leads to new requirements that the customer was not aware of at the beginning. If this happens very late in the project, better to make a feasibility analysis by prioritizing all the remaining tasks and allocating them to the remaining period. If the new requirement is feasible within the timeframe, it is possible to accept it. Otherwise, offer to put it in the future work list.

4.2.4. Demonstration failure during the TNO-ESI symposium

The TNO-ESI symposium is a very important event in this project. Lots of potential end users are visiting the symposium; therefore, the customer wants a concrete demonstration to attract those customers.
Therefore, successful demonstration is critical and to achieve that, it is better to avoid last moment changes in the system. Also, repeated demonstration rehearsal is needed. Furthermore, a backup demonstration environment/laptop is needed in case of any hardware or software failure.

4.2.5. Architecture and design change at last moment

Architecture and design changes might occur at the end of the project because of the insufficient user requirement definition or wrong design decisions. To avoid that, it is better to discuss with the customer and supervisors frequently to have their feedback earlier. Also, discussing with other technical experts will be helpful to avoid making wrong design decisions.

5. System Requirements

Abstract – In this chapter, the detailed system requirements are discussed. The chapter is divided into two parts; functional and non-functional. The section for functional requirements covers high-level functional requirements in this system. The section for non-functional requirements explains all the quality attributes of CoDeX.

5.1 Functional requirements

The following are the high level functional requirements of this project. The detailed design of these requirements is discussed in the architecture and design section.

5.1.1. Define an experiment

The user should be able to define an experiment through a web interface. The experiment definition consists of flows of models and tools. Tools are any kind of executable files such as .exe, .jar, .bat. This experiment definition environment has to be designed and developed from scratch in this project.

5.1.2. Feed the experiment with parameter values

Design Framework should be extended to provide the parameter values for the model. Parameter values can be one value or a set of values.

5.1.3. Set parameter values in experiment

The parameter values provided by DF should be assigned to the corresponding parameters in the Experiment Definition. If the parameter values are one value, they can be assigned to the model parameters directly. If they are a range of values, the algorithm should iterate through the value ranges and decide which value should be assigned to the parameter. In this project, a POOSL model is used. Therefore, POOSL extension is needed to get/set parameter values programmatically.

5.1.4. Run experiment

After feeding the experiment with parameter values, the user should be able to run the experiment via the Experiment Definition interface or DF. The experiment can take a long time to execute and the user might terminate the connection with server by closing his/her web browser. Therefore, the execution of the experiment should not be dependent on the web browser or internet connection. The user should be able to see the experiment run progress next time when he/she opens the Experiment Definition page.

5.1.5. Show run progress status

As mentioned in previous requirement, the user should be able to see the run progress status any time he/she wants during the execution. System should notify the user with all kinds of information related to the execution process such as a progress status or error messages.

5.1.6. Show experiment result

After the experiment is finished running, the final result should be shown on either the DF or Experiment Definition page.

5.2 Non-functional requirements

The terminologies used in the non-functional requirement are chosen based on the "ISO SQuaRe Product Quality" document [12].

5.2.1. Modularity

Modularity refers to how to define minimal interfaces between DF and EE. The main ingredients of this project are still being developed. Therefore, tool dependency should be as low as possible. The tools should be able to work both independently and together with no problem.

5.2.2. Modifiability

Currently, several tools are specified in this project; however, new tools can be added in the future. Therefore, the system should be able to be easily modifiable to adapt to the future possible tools without depending on their technology or architecture.
Since EE aims for support of different models and executables, an important issue is how much effort a developer needs to make when introducing other models and executables. The effort should be as little as possible.

5.2.3. Usability

Usability is related to the 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 following aspects:

- Learnability: even users who are not familiar with the EE tool at all should be able to start a simple use case quickly.
- Operability: the users can accomplish their tasks with minimal effort and no redundant procedures are required.
- User error protection: GUI should be simple and understandable by preventing user from performing wrong actions. Intuitive GUI design and a sufficient User Manual can play an important role in this requirement.
- User interface aesthetics: GUI should look elegant and appealing, which can attract more customers.

5.2.4. Functional appropriateness

The final product has to fit for end user's purpose by satisfying all stated, implied, or hidden needs (Section 5.1). So, to achieve functional appropriateness, the designer should pay attention to the real customer-needs and design and implement a system that can provide the functionalities which can be very useful for end users.

5.2.5. Availability

The Experiment defined by the user can be huge taking several days to execute. Therefore, the system should be reliable by being always available during the long execution.

5.2.6. Portability Compliance

The system should be able to work on both Window and Linux platforms.

5.2.7. Installability

The system should be easily installable with a good installation manual.

6. System Architecture

Abstract – In this chapter, the overall architecture of CoDeX, along with the important decisions and their justifications is covered. The detailed design is discussed in next chapter.

6.1 Introduction

This chapter provides a comprehensive architectural overview of CoDeX. Different architectural views are presented in order to depict different aspects of the design. The four views are: logical, development, process, and physical. In addition, selected use case or scenarios are utilized to illustrate the architecture serving as the "plus one" view. Hence, the architectural view model contains 4+1 views.

Some of the Figures and technology alternative evaluation in Chapter 6 and 7 are also appeared on "Coherent Tool Support for Design-Space Exploration" by Fangyi Shi [13] as a joint work.

This project is a two-person project and each person contributed greatly on certain parts. My personal contributions in this project are application of Client-Server Architecture (6.2.2), Event-Driven Architecture (6.3.3), MVC Architecture (7.2.1), and Dataflow Architecture (7.3.2). Also, technology alternative investigations; PHP vs JSP (7.2.2), Websocket vs Polling (6.3.3), Graph Drawing Libraries (8.2.6) are done by me. I also discovered and used MINI framework (7.2.3), Jetty Web Socket server (7.3.5), jsPlumb library (8.2.6, 8.2.4) in this project. JavaFBP library (8.5) was also discovered during the investigation. Unfortunately, this library was not used because of the time limitation. Database design is also my personal contribution in this project.

6.2 Logical view

The logical view by definition contains the logical components of a system and their interactions. Its purpose is to specify the functional requirements of the system. From design to implementation, the logical view of CoDeX explains more about how the CoDeX components interact with each other in detail.
There are three main logical components of CoDeX, which include Design Framework, Exploration Experiment (EE) components; EE Definition Handler and EE Execution Handler. Design Framework is an already existing component, which needs to work together with EEs components. Therefore, an interface between DF and EE Components has to be defined.

6.2.1. **Context**

The customer wants to have a web interface in the EE component for portability reasons. Web applications can be located on remote servers and provide the same services for all users while they can also be installed on local computer.

The EE component provides two types of services: Experiment Definition and Experiment Execution. Therefore, the EE component is divided into two sub-components: EE Definition Handler and EE Execution Handler.

The EE Definition Handler provides a service to define an experiment and monitor its execution.

The EE Execution Handler provides a service that manages experiment executions. One important characteristic of the experiment is a long execution. The **availability** attribute of this service during the huge automated exploration, which might take several days, has an important role in the overall architecture.

6.2.2. Client – Service Architectural Pattern

Client – Service Pattern [14] is chosen to provide service **availability** during the long execution. The three main components are divided into Client and Server applications, as shown in Figure 5. Design Framework and EE Definition Handlers are the client applications that share the service provided by the EE Execution Handler.
Also, a technology choice of the EE Execution Handler affects availability. The detailed design along with the technology choices are detailed in Section 7.3.

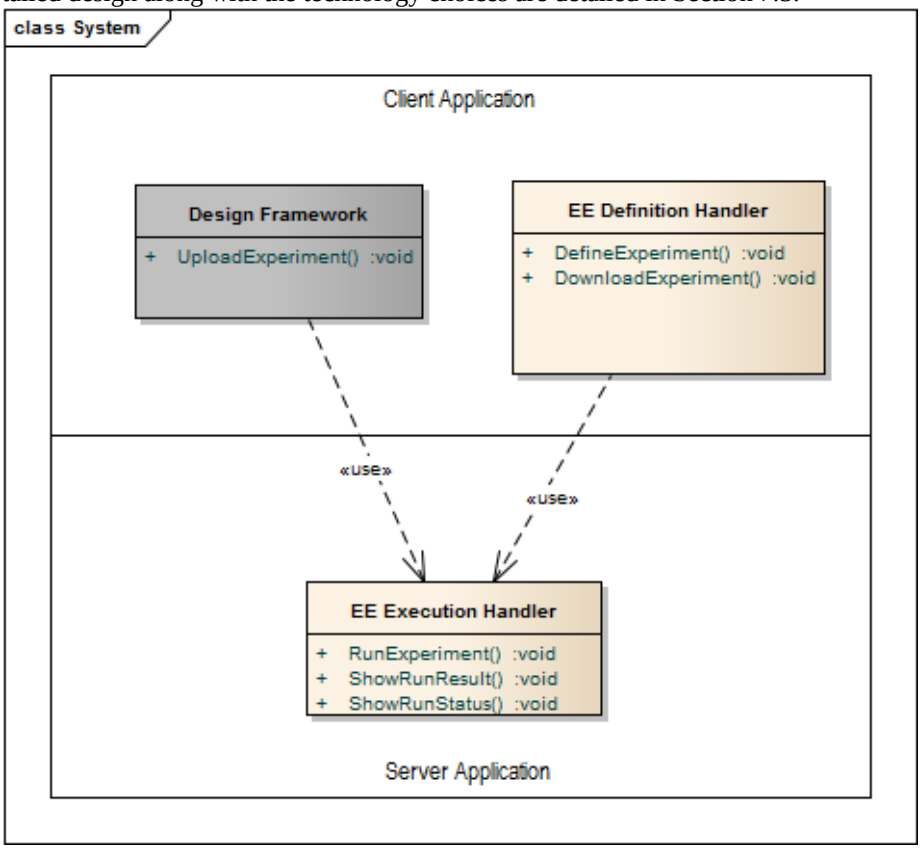

Figure 5 Logical View of CoDeX

6.3 Development view

The rationale for component division is described in the previous section, Logical View. This view focuses on the actual organization of the components and interfaces between them. The design problem and solution for each component are detailed in Chapter 7.

6.3.1. Core Components

The three core components are EE Definition Handler, EE Execution Handler, and DF. The overview of these three core components are in Figure 6.

- **EE Definition Handler:** It is an application to define concrete experiments that deal with the user interfaces and database management. The detailed design of this component is addressed in Section 7.2.
- **EE Execution Handler:** This component focuses on the logical rules of handling all the executions and also manages the database and file system. The detailed design of this component is addressed in Section 7.3.
- **DF:** The implementation of the DF component is part of the DF team's responsibility. However, interfaces between DF and EE need to be well defined in the scope of this project.

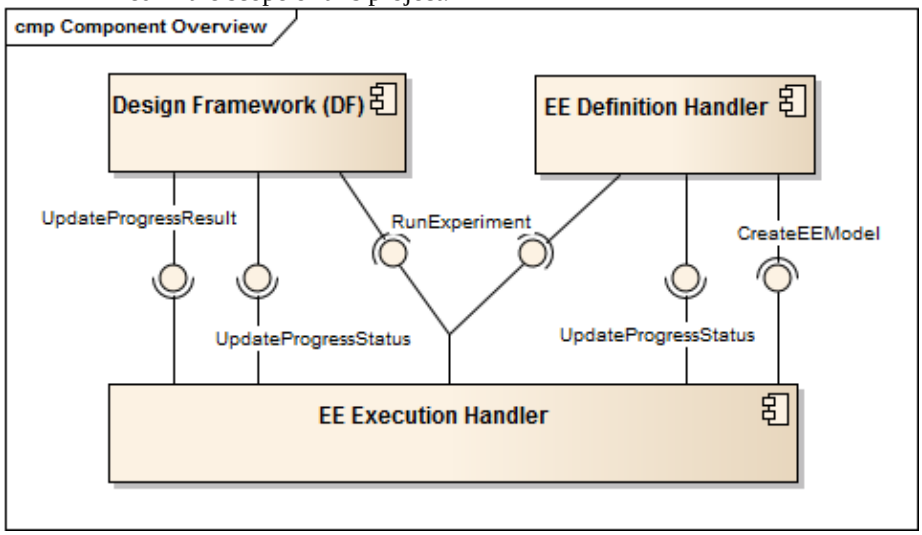

Figure 6 Component overview of CoDeX

Figure 6 shows the interfaces between each component. The components are represented by rectangles. Each component has provided and required interfaces. The interface symbols with a complete circle at their end represent an interface that the component provides. Interface symbols with only a half circle at their end represent an interface that the component requires.

EE Execution Handler communicates with DF and EE Definition Handler separately. Also, there is not direct communication between DF and EE Definition handler for modularity reasons. The communication between them is done via the user, as described below.

6.3.2. Interfaces between DF and EE Definition Handler

This interface plays important role for the ***modularity*** requirement between the DF and EE Definition handler. The DF team does not want a change in DF's current architecture by which DF uploads experiments as a model. Therefore, a special file is needed to represent an experiment defined in EE Definition Handler (Figure 7). This interface is a file that has .ee extension produced by the EE Execution Handler. The user can download this model through the EE Definition Handler and save it into his/her local file system. Then the user can upload this file to DF as a transformation model. An EE Model 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.

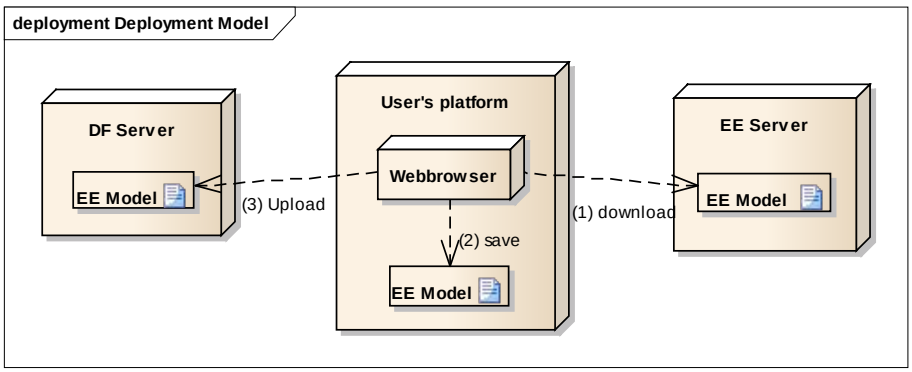

Figure 7 An interface between DF and EE Definition Handler

6.3.3. Interfaces between server (EE Execution Handler) and client applications (EE Definition Handler and DF)

6.3.3.1 Event-Driven Architecture (EDA) vs Service-Oriented Architecture (SOA)

The interface between server and client applications can be implemented in either EDA[15] or SOA[14]. Websocket is the base technology for EDA and traditional Ajax polling is a basic implementation of SOA having Webservice on top of this (Webservice also can be used on top of Websocket). Therefore, understanding how these technologies work and finding out which technology will suit better in our situation are needed.

Table 1 shows the comparison result between polling and websocket technologies [16].

Table 1 Comparison result between Web Socket and Polling

Criteria	Web Socket	Polling
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 service URL for calling back.
Base Technology	TCP	HTTP
Implementation	More complex. Have to implement everything on top of web socket protocol. However there are 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.

Polling uses request-response protocol HTTP. The client submits an HTTP request message to the server and the server returns a response message to the client. If there is not anything to respond, the server just returns an empty message. However, each HTTP message contains a header part which has to be created on every request-response connection. This header creation process affects the performance of the polling mechanism tremendously [16].

There are two key benefits that Web Sockets offer over long polling. First, there is a great reduction in overhead. Long polling needs to send headers for every request and every response. That is not an efficient use of bandwidth. Second, long polling is one-way communication. If something changes on the client whom the server needs to know about, there is no good way to let the server know. With Web Sockets, the communication is bi-directional. The client can talk to the server while the server is talking to the client. With Web Socket, each frame has only 2 bytes of packaging (a 500:1 or even 1000:1 reduction). No latency involved in establishing new TCP connections for each HTTP message [16].

6.3.3.2 Which one is better in our situation? EDA or SOA?

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 a very short time, there are no big differences by using web socket or polling. However, for the executions that require hours or days, the 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, the web socket is chosen for better **performance** and **efficiency**.

The interfaces provided by the server application are:

- **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.
- **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.

The interfaces provided by client applications are:

- **UpdateProgressStatus** is provided by both DF and EE Definition Handler and invoked by EE Execution Handler. It updates the progress status for an experiment execution.
- **UpdateProgressResult** is provided by DF and called by EE Execution Handler. The interface helps to send output data to DF

6.4 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.

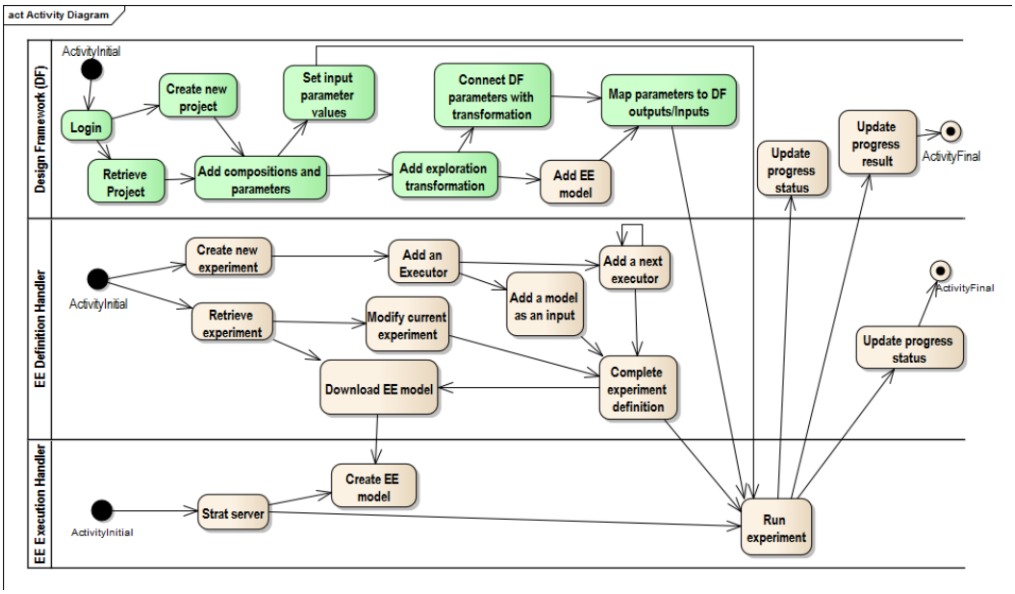

Figure 8 Activity Diagram of CoDeX

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 that cross two different components indicate the interactions be-tween two components.

6.4.1. Activities in DF

In DF, the actions in green have already been implemented. Only three actions are introduced in order to apply an experiment. They require some preparations before adding 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.

6.4.2. 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 2.

Table 2 Activities in EE Definition Handler

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 2 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.

6.4.3. 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* (illustrated by green in Figure 9) action is provided by EE Definition Handler or DF. *Update the final result* (illustrated by green in Figure 9) 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 given in Table 3.

Table 3 Activities in EE Execution Handler

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.

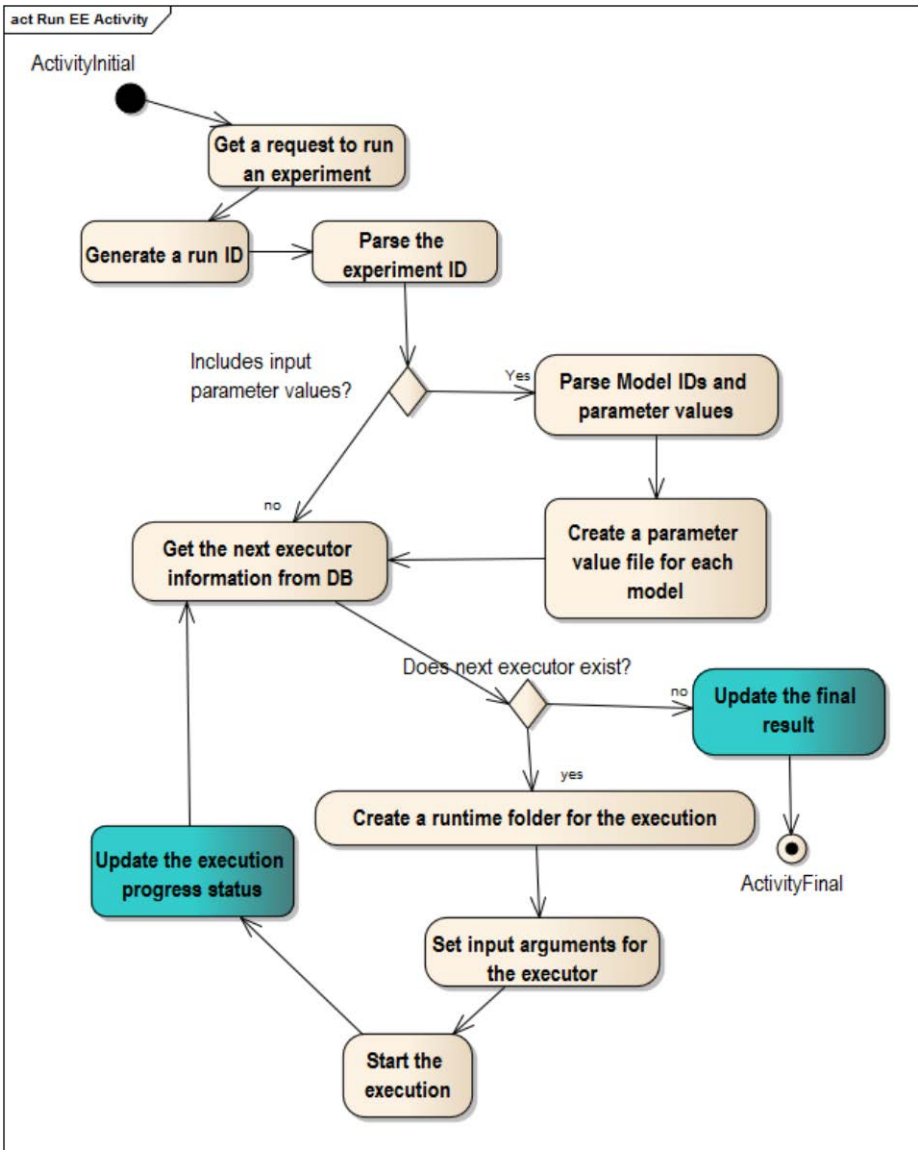

Figure 9 Run experiment activity in EE Execution Handler

6.5 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 server and an execution environment, while the DF part needs a web server and a database server. 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, and 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 to continue with the rest of the tasks at the same time.

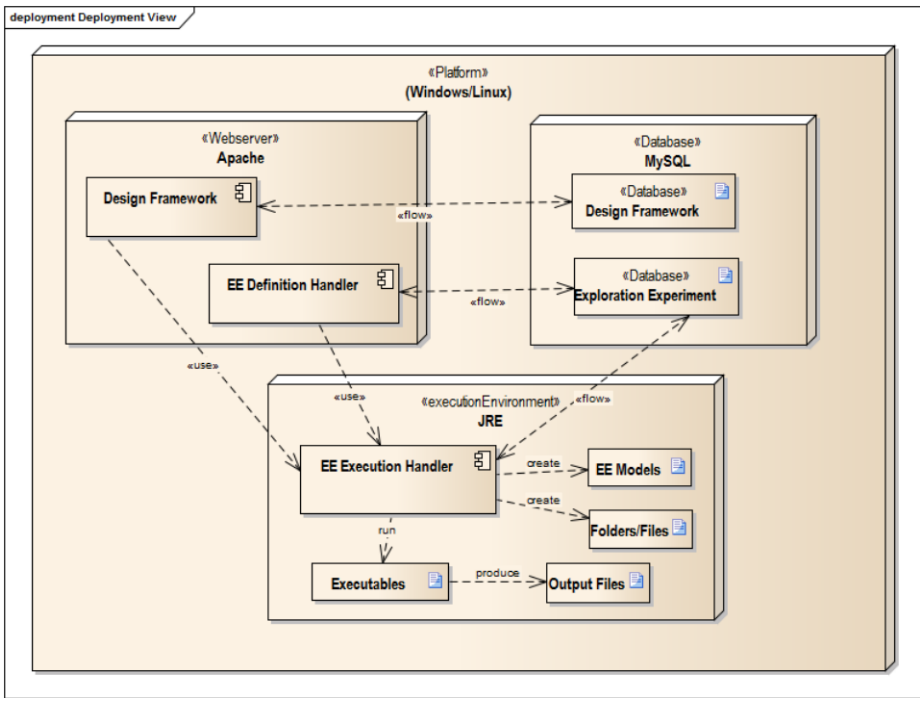

Figure 10 Deployment View of CoDeX on one platform

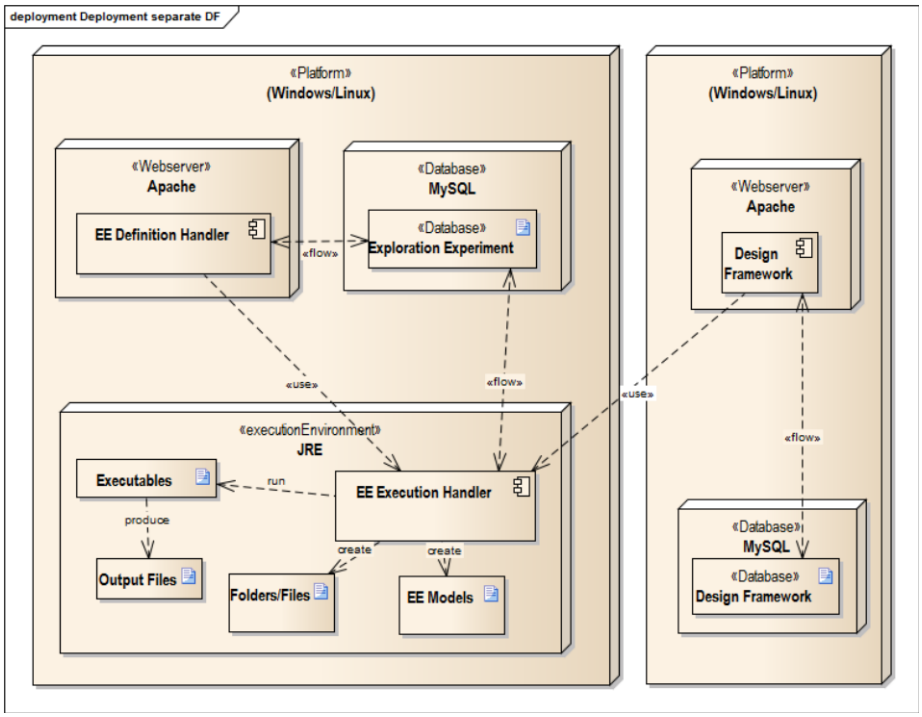

Figure 11 Deployment View of CoDeX on multiple platforms

6.6 *Use case view*

The use-case view captures the functionalities of the CoDeX as seen from the user's point of view. Figure 12 provides an overview of the essential steps a user would follow to create and simulate.

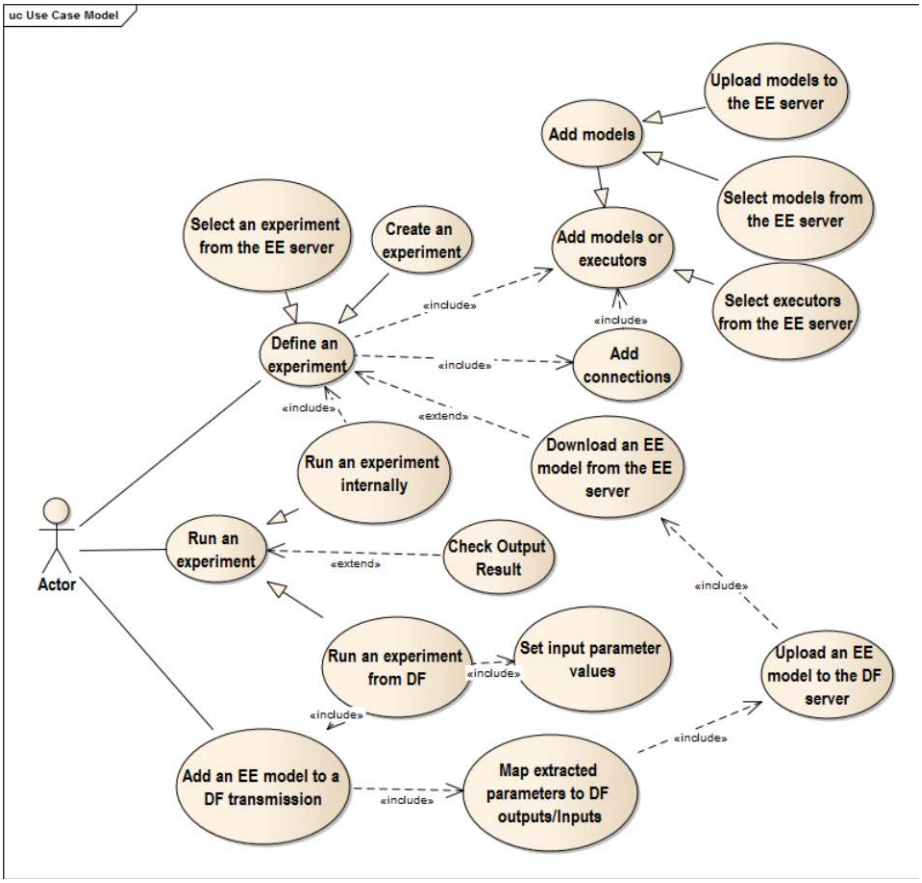

Figure 12 Use Case View of CoDeX

6.6.1. User 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 is configured with 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 Step 5 and Step 7 can be reversed.
10: Step 5 and Step 7 are 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.

6.6.2. 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.	

6.6.3. Use case 3: Run an experiment from EE

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 from EE Defini-

	tion Handler 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.	

6.6.4. Use case 4: 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.	

7. System Design

Abstract – So far, the overall architecture choices are detailed, but not the design of each component itself. This section covers the detailed design of each component and the design decisions separately.

7.1 The Database

The database is an important part of the application that contains all the information and application logics. The EE Definition Handler and EE Execution Handler have different DB Handler; however, both handlers manipulate the same database tables specified below (Figure 13). The reason for having two separate DB Handlers is **modularity**. Also, two DB Handlers function differently. See Section 7.2.5 and 7.3.3 for more details.

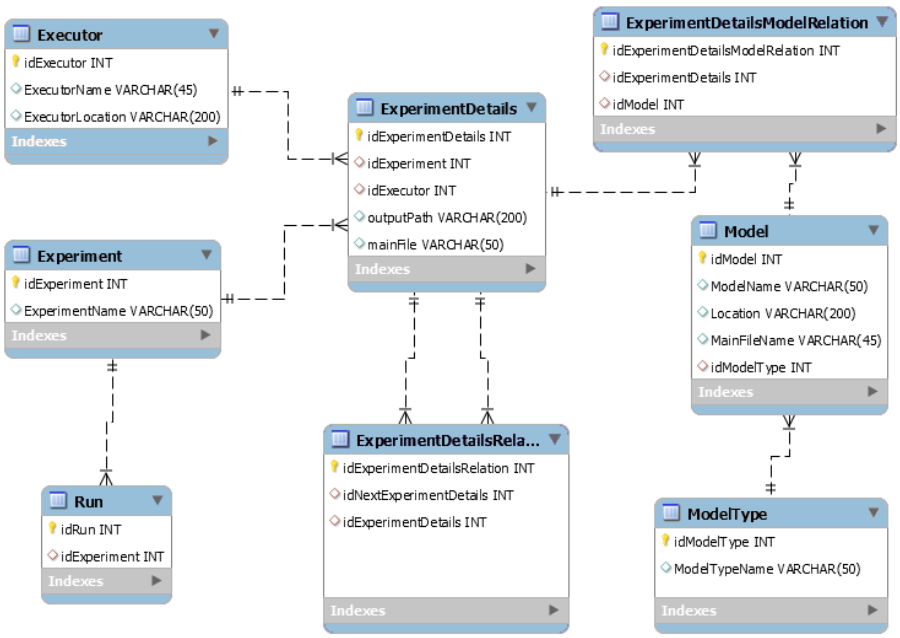

Figure 13 MySQL Database tables

The following is the explanation of each table:

- The Experiment table contains experiment information. The table has the following two columns:
  - idExperiment – primary key of the table
  - ExperimentName – a column for experiment name

- The Executor table contains the executable's information. The table has following columns:
  - idExecutor - primary key of the table
  - ExecutorName – name of the executor
  - ExecutorLocation – physical location of the executable file

- The ExperimentDetails table contains detailed information of executors used in the experiment. The table has following columns:
  - idExperimentDetails - primary key of the table

  - idExperiment – reference column to the idExperiment column of the Experiment table. One experiment can have multiple experiment details
  - idExecutor – reference column to idExecutorColumn of the Executor table. One experiment detail corresponds to only one executor.
  - outputPath – physical path for the execution result
  - mainFile – name of the main file of the execution result

- Model table contains information of the models. The table has the following columns:
  - idModel - primary key of the table
  - ModelName – name of model
  - Location – physical location of a model
  - MainFileName – main file name of the model (a model can consist of a set of files and folders; however, there should be one main file within that set. Also, mainfilename can be blank if it is not needed for the execution)
  - idModelType – reference column to idModelType column of ModelTyles table. Each model should be referred to one of the model types

- ModelTypes table contains information of all model types.
  - The table contains following columns:
    - idModelType - primary key of the table
    - ModelTypeName – name of the model type
  - Currently, following model types exist in a database
    - POOSL – files with .poosl extension
    - TXT – files with .txt extension
    - IMG – any kinds of image files
    - XML – files with .xml extension
    - FOLDER - folders
    - TEXTVALUE – text values (some executables produces only text values)
  - Adding a new model type to the system is not only restricted to the information of this table. A few other changes are necessary. Detailed instruction is a specified in separate document called Development Documentation.

- Run table contains run information. The table has the following columns:
  - idRun - primary key of the table
  - idExperiment – a reference column to the idExperiment column of the table Experiment. One experiment can have multiple runs.

- ExperimentDetailsModelRelation table contains the relationship between experiment details and models. The table contains the following columns:
  - idExperimentDetailsModelRelation - primary key of the table
  - idExperimentDetails – reference column to the idExperimentDetails column of table ExperimentDetails.
  - idModel – reference column to the idModel column of table Model

- ExperimentDetailsRelation table contains the relationship between the experiment details. In other words, this table contains the connection chain of all the tools (executors) used in the experiment. The table contains the following information:
  - idExperimentDetailsRelation - primary key of the table
  - idNextExperimentDetails – a reference column to idExperimentDetails of the table ExperimentDetails. This is the ID of the experiment details on right side of the connection.
  - idExperimentDetails – a reference column to idExperimentDetails of the table ExperimentDetails. This is the ID of the experiment details on left side of the connection.

7.2 EE Definition Handler

Definition Handler is an interactive web application that helps the user to define the experiment and monitor its execution.

7.2.1. Sub-components and MVC Pattern

GUI part is typically the most frequently modified portion of an interaction applications. For this reason, it is important to keep modifications to the GUI part separate from the rest of the system. Therefore, modifiability is an important quality attribute for this component. To achieve **modifiability** quality attribute, Model-View-Controller pattern [14] is applied in this component. MVC pattern separates application functionality into three kinds of components (Figure 14):

- A model – contains application data. DB Tables component belongs to this view.
- A view – displays the data and interacts with the user. The GUI component belongs to this view.
- A controller – mediates between the model and the view and manages the notifications of state changes. The following three components belong to this view:
    - GUI Handler: a component that handles GUI
    - Definition Coordinator: a mediator component between GUI Handler and DB Handler
    - DB Handler: a component that handles DB Tables

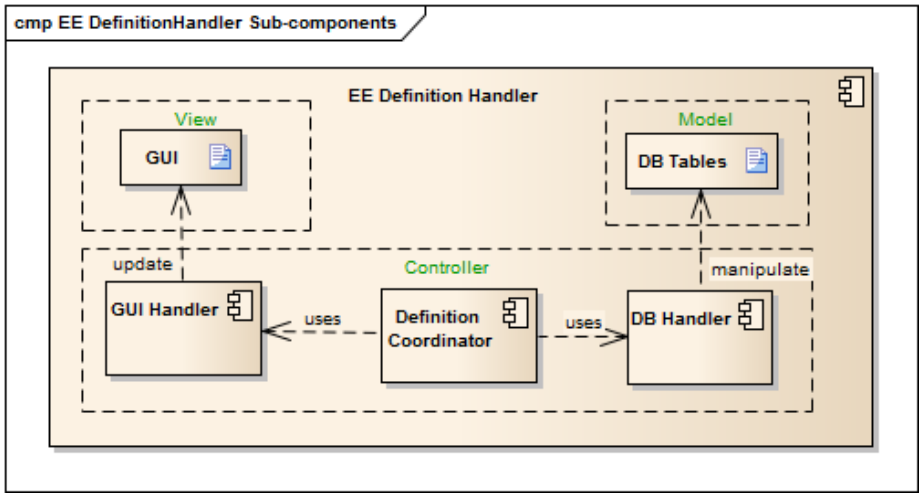

Figure 14 EE Definition Handler Sub-components

7.2.2. Server language

The two most popular and open source server languages for web application development are PHP and JSP. To make a choice between them, we made following comparison research (Table 4).

Table 4 Server language comparison result

Criteria	PHP	JSP
Usage	***Popular. There are lots of 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 available, such as Eclipse, Netbeans
Learning Curve	***Easy to learn***	Difficult to learn
Performance	Slow	Fast. 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.

7.2.3. MVC – PHP Framework

Building software applications from scratch can be a complex and time consuming process; however, utilizing a framework can help to develop it faster (by reusing generic components and modules), and work better. Using a framework also facilitates scalability and long-term maintenance by complying with development standards, keeping the code organized and allowing the application to evolve and grow over time.
Therefore, a ready-to-use framework MINI [17] is chosen in this project because of the following advantages:

- Extremely simple
- Highly documented
- Lightweight
- Easy to learn (native PHP code)
- Promotes the basic php-coding standard (PSR 1/2 coding guidelines [18]) and usage of PHP Data Objects (PDO[19])

The other alternative to this framework is CakePHP [20] which is more popular than MINI but the fact that it is simple, learnable, and lightweight is not comparable with MINI.

7.2.4. Web Development Platform

To achieve **installability** quality attribute, XAMPP [21] is used in the development of Definition Handler. To develop a web application, webserver (Apache), server language (PHP), and database (MySQL) need to be installed separately. However, with XAMPP all these installments is done at once, which makes whole installation of the application much easier.
The installation guide with XAMPP is provided in the separate document named "Installation Manual."

7.2.5. DB Handler

DB Handler of EE Definition Handler has the following sub-components (different php files):

- ModelModel: a component manipulates table Model. The component has the following functions:
  - AddModel – inserts a new model information
  - UpdateModelInfo – updates the existing model information
  - UpdateModelDir – updates model path
  - RetrieveModelTypes – selects all models types
- ExecutorModel: a component manipulates table Executor. The component has the following functions.

    - CreateExecutor - adds new executor information
    - RetrieveAllExecutor – retrieves all executor information
- EEModel: a component manipulates table Experiment. The component has the following functions.
    - CreateEE – inserts a new experiment information
    - RetrieveAllEE – selects all experiment information
    - RetrieveAllRuns – selects all run information
    - RetrieveAllModels – selects all model information
    - RetrieveEDs – selects all experiment details
    - AddExecutor – adds new experiment detail for the experiment

- EDModel: a component manipulates ExperimentDetails information. The component has following functions.
    - CreateED – creates new experiment detail
    - ConnectNextExecutor – connects experiment detail for another experiment detail
    - RetrieveAllED – retrieves all experiment detail

7.3 EE Execution Handler

This section covers details of the architecture and design issues of Execution Handler.

7.3.1. Sub-components

The EE Execution Handler component consists of following sub-components (Figure 15):

- Executable Coordinator: main coordinator component of the entire execution
- Execution Handler: a component that handles a single execution
- Update Handler: a component that handles update status
- DB Handler: handles MySQL tables
- File Handler: handles ee Model and folders/files

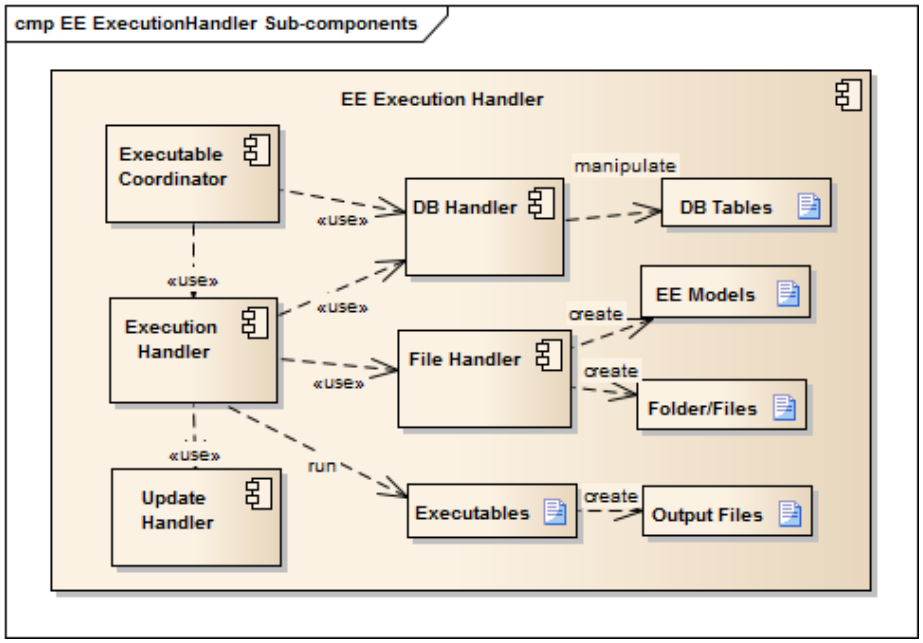

Figure 15 EE Execution Handler Sub-components

7.3.2. How to achieve modifiability? Pipeline versus Dataflow

One of the main requirements in this project is ***modifiability***. The EE Execution Handler has to be ***modifiable*** with new extensions. The new extension can be any kind of executable files such as .exe, .jar, and .bat. To achieve an easily modifiable system, we needed to analyze the main ingredients, executable files, and generalize them. The following are the results of the generalization analysis:

- Executables can be executed with a system call command in any OS
- Executables take input arguments: executables can take any number of inputs. Inputs can be any kind of texts formats.
- Executables convert the input data to output data. Executables can produce any kind of files/folders as outputs.

The conclusion based on the above analysis is that each executable is an operation (blackbox) that takes input data and produces converted output.

There are two architectural patterns can be used in the above situations: Pipeline [14] and Dataflow.

The Pipeline and filter pattern is used to transform a stream of discrete data items from input to output. The system is divided into reusable, loosely coupled components with simple, generic interaction mechanism. The data arrives at a filter's input ports and transformed data is passed to the next filter via the current filter's output port.

In this sense, the pipeline and filter patterns perfectly match in our situation. However, this pattern does not allow cyclic connection. Therefore, it is impossible to achieve an automated DSE (Section 2.5) with simple Algorithm (Section 8.6.2) which requires a loop back connection. Also, in pipeline and filter pattern, all the filters run in parallel but in in our case, executables cannot be executed without input data. Therefore, this pattern does not fully satisfy our needs.

Another candidate pattern is a Dalaflow pattern [22][23]. Dataflow is similar to pipeline because it is also a stream of data transformation. The difference with Pipeline is, the connection can be in a structure of a directed graph. Therefore, a cyclic connection is possible. Also, all the components do not need to run at the same time.

Therefore, the dataflow pattern is chosen in the implementation of EE Execution Handler.

7.3.3. Logical view

Figure 16 shows the class diagram of EE Execution Handler, which reflects the Dataflow architectural pattern. 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* (creates a dataflow chain), which provides concrete functions about one executor's execution, such as creating output path in the runtime or setting the inputs for one execution. Every *ExecutionDetail* contains an abstract class *Executor* (main operations. As an abstract class, *Executor* is specialized by four concrete classes, *PPOOSL, Poosl2XML, Rotalumis, and 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 thatneeds 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. *DBHandler* 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 is 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*.

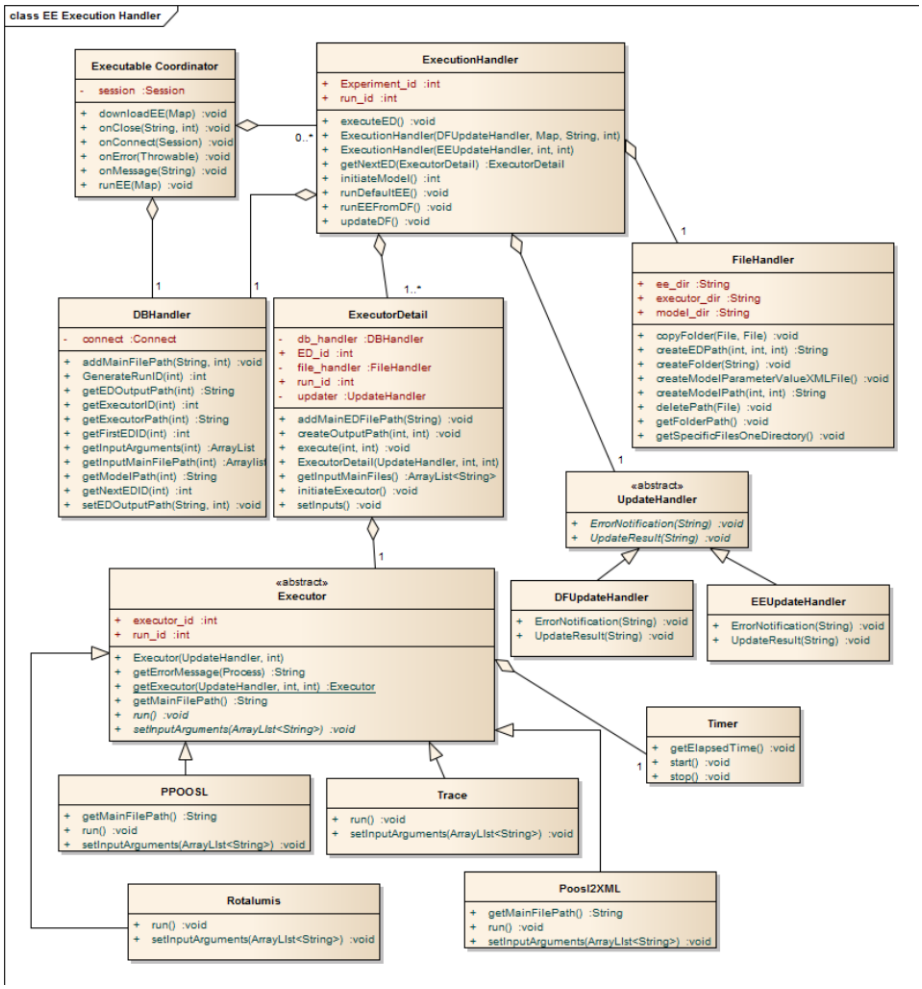

Figure 16 Class Diagram of EE Execution Handler

7.3.4. 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.
- JAR files executables, which are implemented in Java.

C++ is another alternative, which is much faster than JAVA, but was not chosen in this project because of the learning curve. The main goal of this project is a quick prototype; therefore, it was better to avoid risks of delay because of the lack of experience with a certain technology.

7.3.5. The Jetty Web Socket server

An environment that can help to integrate Web application, the EE Definition Handler, to Java application, EE Execution Handler, through Websocket interface (see Section 6.3.3 for more details) is needed. The Jetty Websocket server [24] was chosen for this purpose because it is popular, powerful, and well-documented.

7.3.6. Deployment Diagram of EE Execution Handler

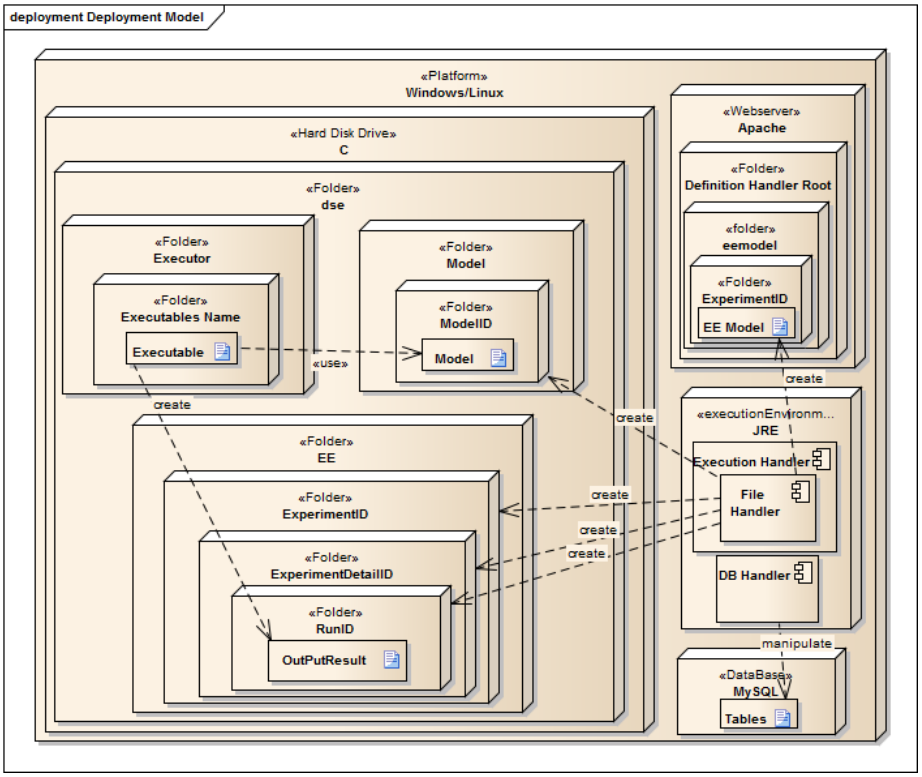

Figure 17 Deployment diagram of EE Execution Handler

Figure 17 shows all the files and folder structures deployed in the server's platform. The EE Execution Handler is in charge of handling all files and folders and data information.

The following four kinds of files are used in the system. Each file is located in different folders:

- *Model* is any file or set of files and folders that describes the model structure. Model files are stored under following path: c://dse/Model/<<ModelID>>/
- *Executable* is the executable versions of the tools available in the system. The location of this file is: c://dse/Executor/<<Executable name>>/
- *Output Result* is any kind of files or set of files and folders produced by the executor. The location of these files are: c://dse/EE/<<ExperimentID>>/<<ExperimentDetailID>>/<<RunID>>/
- *EE Model* is the special kind of model that is produced by Definition Handler and can be downloaded from EE Definition Handler. This is the main interface between EE Definition Handler and Design Framework. These files are located under the eemodel folder inside the Apache server's root directory because user accesses these files through the EE Definition Handler. (See Section 6.3.2 for more details)

8. Implementation

Abstract – In this chapter, the implementation phase is detailed. Emphasis is put on describing the new artifacts that need to be designed for this phase, including their requirements and not the mere implementation of design outlined in Chapter 6 and 7.

8.1 Introduction and Individual Scope

Although the ideal scenario would be that the implementation phase is merely a simple step, the reality is always far more complicated. In this project, two prototypes are implemented. First prototype concentrated on the Experiment Definition with one single flow with no loop back connection. The implementation was built together with Fangyi Shi. The following are the components included in the individual scope of the first prototype (Figure 18).

- EE Definition Handler
  - Definition Coordinator – database related parts
  - DB Handler
- EE Execution Handler
  - DB Handler
  - File Handler

The second prototype was completely implemented in the scope of this report. During the implementation of first prototype, several new requirements came up; therefore, a new GUI was introduced in the second prototype. Functional requirements of Check Run Progress and Check Run Result were implemented in second prototype. Also, an implementation challenge of a Loop Back Connection during the second prototyping led to the design change in EE Execution Handler.

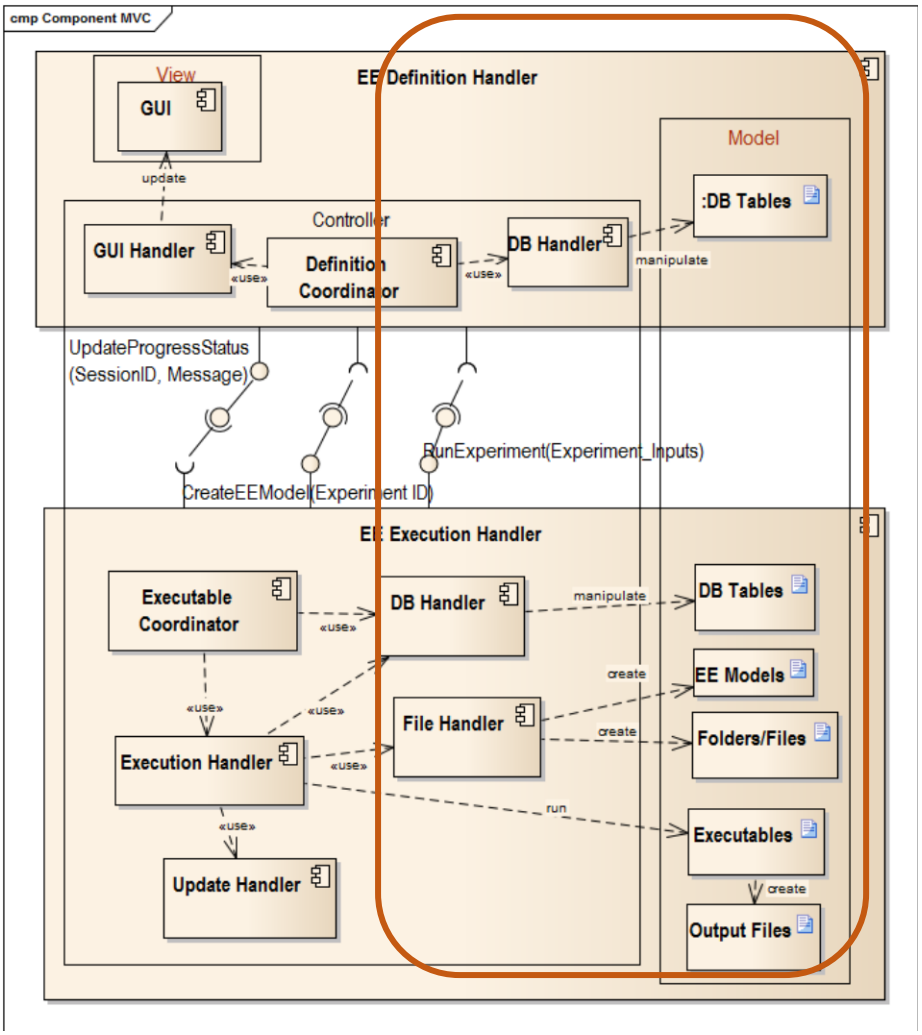

Figure 18 Individual Scope in first prototype

8.2 GUI Improvement

8.2.1. GUI of the first prototype

Figure 19 illustrates the GUI of experiment definition in first prototype. Blue rectangles represent Executables (with the id over it) and pink rectangles represent Models. User has to right click on the executable to connect a model to it or connect it to another executable. This prototype helped the customer further analyze his needs and introduced new requirements.

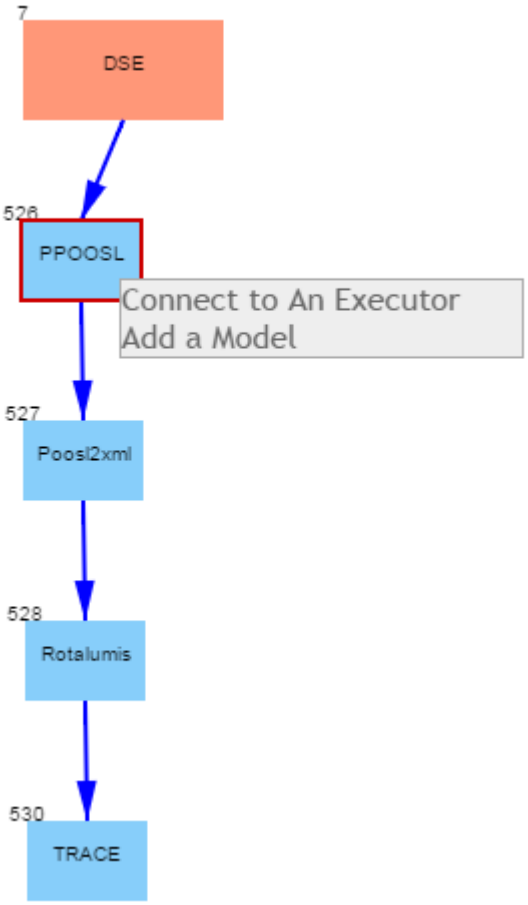

Figure 19 GUI of first prototype

8.2.2. New functional requirements

Following functional requirements are added to make the prototype more complete and appealing for the user:

- Different way to make connection: The way to make connection is not so intuitive in the first prototype. Also, user cannot connect to the existing executor; therefore loop back connection is impossible
- Input/output compatibility check: Output of one executable can be an input of another executable. These output and input types should match to avoid run-time errors in the systems.
- Delete functionality: User should be able to remove items from the GUI in case he/she wants to make change.

8.2.3. A new non-functional requirement

- User interface aesthetics: An appealing GUI is needed to attract customers.

8.2.4. A new GUI in second prototype

Figure 20 illustrates the new version of GUI. The GUI provides following features to solve the restriction in first version of GUI:

- Drag and drop for creation
  - To add new item to the experiment
  - To connect the items in the experiment
- Double click for deletion
  - To remove item in the experiment
  - To remove connection between two items

- Multiple input/output ports
  - o - solid circle represents output ports
  - o - empty circle represents input ports
- Color distinction for output/input compatibility
  - o Color represents the different types of input/output. Only same colored ports can be connected. This helps **to achieve user error protection requirement** by preventing user from performing wrong connection.
- Color representation of model types. Only POOSL and Values types of models are shown in Figure 20.
  - o - folders
  - o - any find of images
  - o - POOSL models
  - o - parameter values. System can have only one Values type of model. This model is used to receive parameter values from DF. When EE Execution Handler receives parameter values, it can find a model that has Values type and update its content with values received from DF.
  - o - text files
  - o - xml files

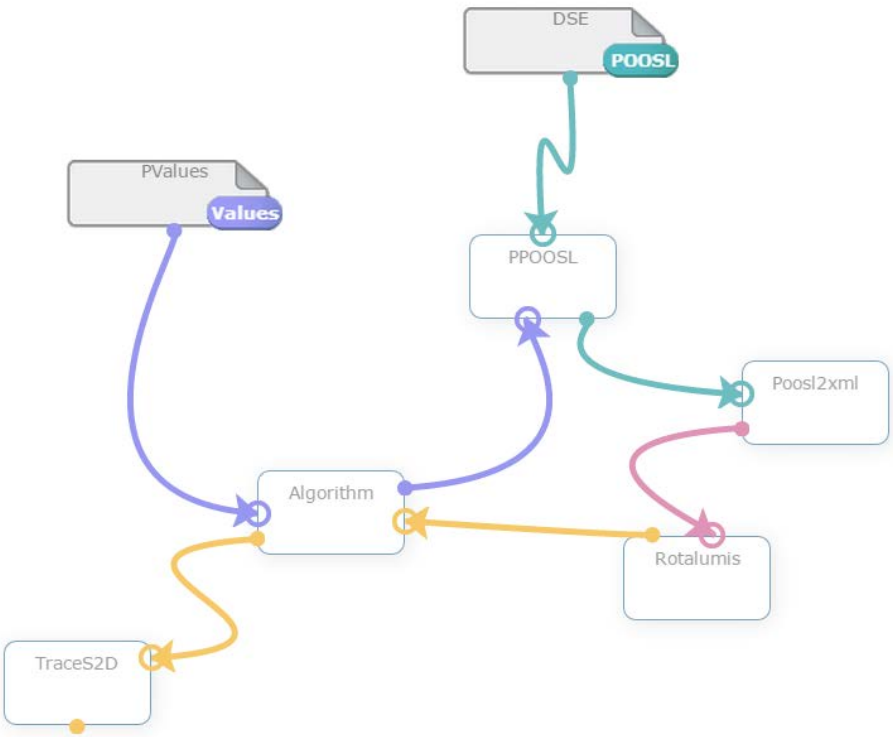

Figure 20 GUI of second prototype

8.2.5. Database change

To achieve above design, a few changes were made in DB Tables.

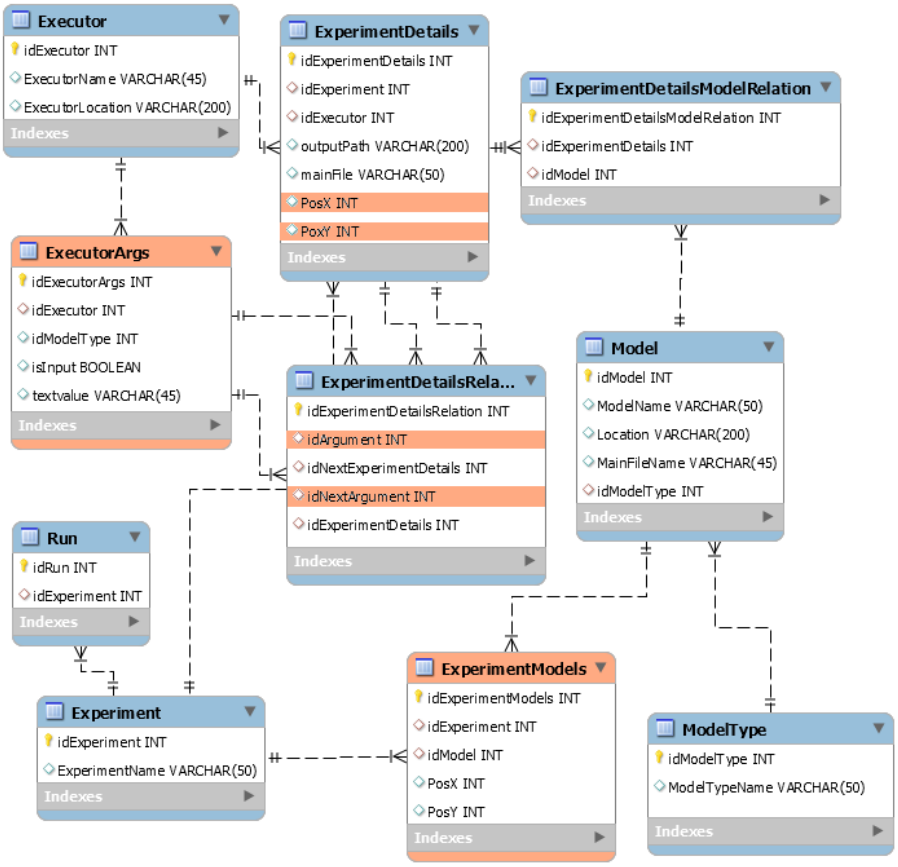

Figure 21 Database change in second prototype

The orange parts in Figure 21 show the changes that occurred in DB Tables because of the new GUI design. Following two tables were added:

- ExecutorArgs: a table contains all input and output arguments of each executor. This table plays important role in input/output compatibility check to obtain user **error projection requirement**. The table contains following columns:
    - idExecutorArgs – primary key of the table
    - idExecutor – reference column to idExecutor column of table Executor
    - idModelType – reference column to idModelType column of table ModelType. It helps to define a model type of that certain argument. Based on this information, a model type constraint check is made.
    - isInput – a column which specifies if the argument is input. The values are set to “true” if it is an input argument or set to “false” if it is an output argument.
    - Textvalue – any text value or comment for the argument.
- ExperimentModels: this table contains information about models used in the experiment. In previous design, each model was directly connected to the experiment details because models could be added by right clicking on the experiment details. However, in new design user can drag and drop the model to add it to the experiment. Therefore, the model has to be connected with the experiment first, then can be connected to experiment details inside the experiment. The table contains following columns:
    - idExperimentModels – primary key of the table
    - idExperiment – reference column to idExperiment column of table Experiment

    - idModel – reference column to idModel column of table Model
    - PosX and PoxY – Model's position values on the screen. The position change can be saved in the database. With help of this, user can see his/her experiment definition items exactly where they were specified.

Following tables were changed:

- ExperimentDetails:
    - PosX and PosY – ExperimentDetails's position value in the screen.
- ExperimentDetailsRelation:
    - idArgument and idNextArgument – these columns save two connected items' argument IDs. All the executables can have multiple input/output ports (arguments). Therefore, it is important to specify which arguments are connected together.

8.2.6. Graph drawing library

A better way to draw graph is needed to achieve User Interface Aesthetic requirement. There are plenty of ready to use javascript libraries which can be used for this purpose. Following comparison study (Table 5) was made to find a good candidate for this project.

Table 5 Graph library comparison result

Library	User community	Easiness	License
JointJS	Small	Well documented	Open
Diagramo	Medium	Not enough documentation	Open
GoJS	Large	Well documented	Commercial
jsPlumb	Large	With plenty of examples and documentation	Open

JointJS [26] and Diagramo [27] are good candidates but JointJS is not so popular yet and Diagramo is not so flexible. GoJS [28] is another powerful but commercial library. jsPlumb [25] library was chosen for its popularity and powerfulness. It is an open source javascript library, which helps to create flow diagram modeling on web application.

8.3 Design change in EE Execution Handler

One of the important characteristics of Dataflow architecture is multiple output ports per executable. In initial design of EE Execution Handler, one executable had only one output port. That means, the executable can produce only one type of output result. If the model type of the executable is POOSL then the executable always has to produce a POOSL as an output result. However, in some case the executables need to produce different types of outputs. This is especially related to the executable, Algorithm (Section 8.6.2) which helps to achieve automated DSE (Section 2.2). The algorithm needs to produce two kinds of outputs; an execution result folder for Trace and an XML file which contains parameter values for POOSL input. In Figure 22, the Algorithm cannot be connected to Trace because it does not have the corresponding output port. Therefore, multiple output port is enabled, which help the database changes specified in Section 8.2.5.

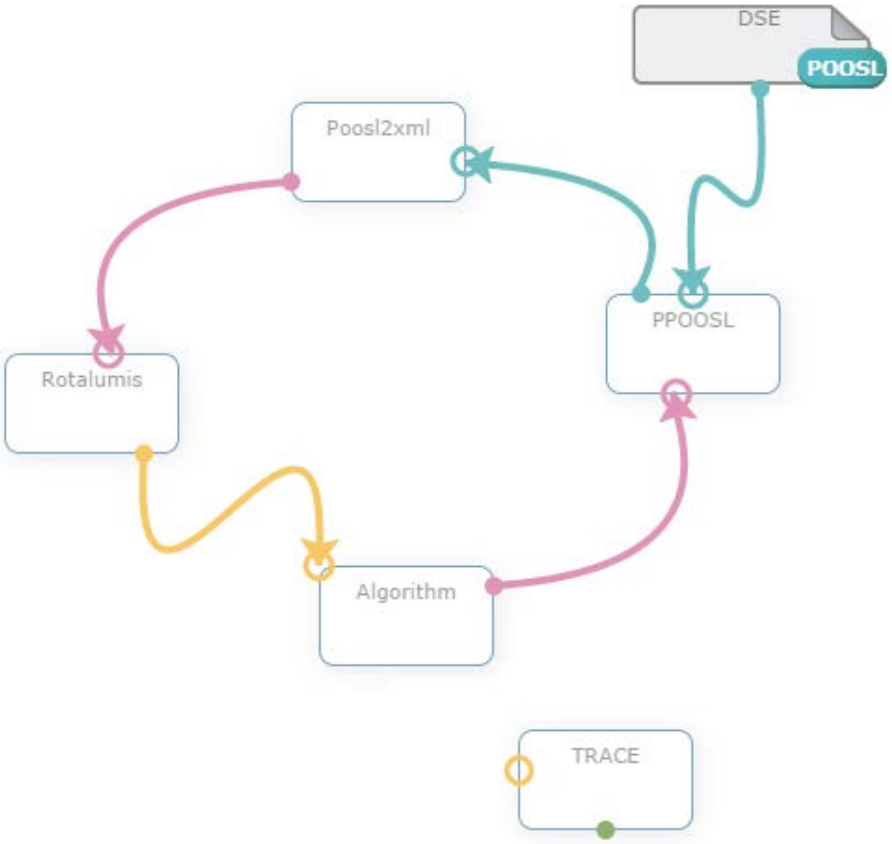

Figure 22 A cyclic connections between the executables with one output port

In Figure 20, the executable named algorithm has two output ports; pink and yellow. The pink port is connected back to the PPOOSL executable that creates a loop in the diagram. The yellow port is connected to the Trace executable. So, based on the result of the execution, Algorithm can decide which output port to enable. If it enables the pink port, the whole circle will be executed again. If it enables the yellow port, the entire execution will finish after Trace's execution.

With help of this change, the loop back connection is enabled; therefore, a system became able to conduct the automated DSE. The real demonstration of an automated DSE with an example (Section 2.5) was conducted to validate this design.

Executables also should be able to enable multiple output ports which leads to multiple flows. In the scope of this project, only one output port can be enabled at the time which means both Trace and PPOOSL cannot be executed at the same time after Algorithm's execution in Figure 20. To achieve that, parallel processing functionality is needed in addition to the current design.

8.4 Design Change in EE Definition Handler

The following functions were added in EE Definition handler regarding the new requirements:

- ExecutorModel:
  - RetrieveExecutorArgs – retrieves executor arguments
- EEModel:
  - AddModel – adds a model to the experiment
  - SetInitialED – sets initial experiment detail. There can be only one initial experiment detail
  - CheckInitialED – checks if initial experiment detail is specified

    - DeleteModel – removes model from experiment
    - ChangeEEModelPosition – changes a model position
- EDModel:
    - ChangeEDPosition – changes the position of the experiment detail
    - AddModel – connects a model to experiment detail
    - ConnectED2MD – connects experiment detail to model
    - RetrieveEDArgs – retrieves all arguments of experiment detail
    - RetrieveFreeArgs – retrieve all arguments of experiment details that are not connected with other experiment details or models
    - DisconnectEDs – remove the connection between two experiment details
    - DisconnectEDModel – removes the connection between experiment detail and a model
    - DeleteED – removes experiment detail from the experiment

8.5 FBP: Suggestion for future improvements

Flow-Based programming (FBP) [29] is a programming paradigm based on the Dataflow Architecture pattern. The FBP defines an application as networks of "black box" processes, which exchange data across predefined connections by message passing. FBP provides solutions for the following issues:

- Parallel processing
- Multiple input/output ports
- Complex and cyclic connections

To achieve FBP in Execution Handler, a ready to use Java library, JavaFBP[30] can be used in the Execution Handler in future work.
A javascript implementation of FBP, NoFlo [31], can be used in EE Definition Handler improvement. NoFlo, a powerful library, can be used to run full flow-based applications with complex dataflow with asynchronous processes. The graph editor layer, which helps to create dataflow chain via the NoFlo's GUI is compatible with JavaFBP.

8.6 New executables

To achieve a full demonstration, two new executables, PPOOSL and Algorithm are implemented in this project. Trace for Gantt Chart and DS Graphs were developed by Fangyi Shi. POOSL2XML and Rotamlumis are being developed by the POOSL team.

8.6.1. PPOOSL

PPOOSL is a POOSL conversion tool that has the following two functionalities:

- Get parameter values: EE Framework extracts parameter names from the POOSL model and list them in the EEModel file; thus, the user of the Design Framework can work with the same parameter values specified in the POOSL model.
- Set parameter values: the parameter values coming from the Design Framework need to be assigned to the POOSL model before execution.

8.6.2. Algorithm

The Algorithm is a simple algorithm, which reflects the design change detailed in Section 9.3. This tool is implemented to demonstrate the DSE example with a range of values. It takes the range of parameter values and assigns each value one by one during iterations of the execution. After finished executing all the values, the Algorithm enables its yellow port and Trace summarizes the results of all the iteration (Figure 23).

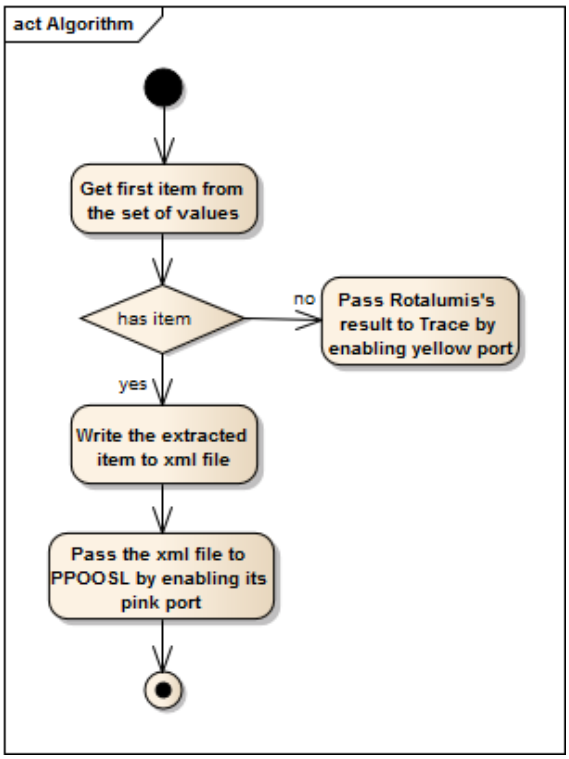

Figure 23 A Simple algorithm for automated DSE

9.Verification & Validation

Abstract – The architecture and design presented in the previous chapter should fulfil all the requirements introduced in Section 5. To ensure that, the verification and validation were applied to the EE Framework. This chapter describes various tests and results for verification and concludes with the validation by answering the question "Is this the right product for DSE?"

9.1 Verification

Since the project goal is a proof of concept, the main effort was not spent on the complete testing strategy. However, to provide a stable prototype, the functional test was conducted manually through the second version of EE Definition Handler and DF.

9.1.1. Functional Test

The test covered following four cases:

- Define an experiment (Table 6): This test case can be done only on EE Definition Handler. User can create new experiment, retrieve previously defined experiment, and download the experiment.
- Feed the experiment with parameter values (Table 7): Feeding the experiment with parameter values can be done both via DF and EE Definition Handler, however, defining through DF is covered by the test case.
- Check the parameter values setting in the experiment (Table 8): The parameter value setting functionality is done by EE Execution Handler. To check the result, user need to see the execution result of PPOOSL from EE Definition Handler as specified in Table 8.
- Run experiment, Show run progress Status, and Show experiment result (Table 9): User can run the experiment, check the run progress status via both DF and EE Definition Handler. Experiment results can be shown only of EE Definition Handler.

The test cases consist of several functionalities and corresponding user activities, expected results toward the activity and actual test results.

Table 6 Test case: Define an experiment

Functionality	*User activity*	*Expected Result*	*Test Result*
Create new experiment	Press "Create" button	"Create new experiment" dialog pops up	Passed
	Enter experiment name and press "OK" button	Pop up dialog disappears	Passed
		New experiment name shows up on top of the page	Passed
		Canvas is cleaned	Passed
Add new model	Press "+" button on the top of the "Models" list	"Add a new model" dialog pops up	Passed
	Choose a model file, fill in "Main file name" and "Model Name", select a model type from the dropdown list	The new model information is shown below the "Upload" button	Passed

	and press “Upload” button		
Add model to the canvas	Drag the model to add from the right menu, Models, and drop it on the canvas	Model shows on the canvas and disappears from the Models menu	Passed
Add executor to the canvas	Drag the executor to add from the left menu, Executables, and drop it on the canvas		Passed
Connecting source and target items (source items can be either model or executor. Target item can only executor)	Press the output port of the source item and drag toward the same colored input port of the target item and drop over	The established connection will be shown by the arrow	Passed
	Press the output port of the source item and drag toward the different colored input port of the target item and drop over		Passed
	Press the output port of the source item and drag toward the any output port of the target item and drop over		Passed
Retrieve an experiment	Press “Retrieve” button	“Retrieve an experiment” dialog pops up	Passed
	Select an experiment to retrieve from the dropdown list and press “OK” button	Experiment name shows on top and all the experiment details defined previously has be shown on the canvas	Passed
Download the experiment	Press on “Download” button	File browser shows up asking where to save the EE Model.	Passed
	Select a path and press OK button	EE Model is saved under the specified path with Model.ee name	Passed

Table 7 Test case: Feed the experiment with parameter values

Functionality	*User activity*	*Expected result*	*Test result*
Upload the EE Model	Double click on the transformation	Transformation details dialog pops up	Passed
	Press “Edit current model” button	Edit an existing model dialog pops up	Passed
	Choose the Model.ee file downloaded via Experiment page and press OK	Model updated message appears	Passed
Set parameter val-	Double click on	Parameter details dialog	Passed

ues on DF	parameter	pops up	
	Insert parameter value in textbox for value and press OK	Parameter saved message appears	Passed

Table 8 Test case: Check parameter values setting in the experiment

Functionality	*User activity*	*Expected result*	*Test result*
Check if parameter values are set in the experiment	Refresh the EE page, retrieve the experiment used in DF, and select latest run	Shows run progress status	Passed
	Press on the "Run Result" over PPOOSL after it finished executing	Run result dialog pops up	Passed
	Press on main.poosl file	Main.poosl file is shown on new tab and parameter values should the parameter values specified in DF	Passed

Table 9 Test case: Run experiment, Show run progress Status, and Show experiment result

Functionality	*User activity*	*Expected result*	*Test result*
Run experiment from DF	Set all parameter values	Run command is sent automatically to EE Execution Handler.	Passed
		The image changes to hourglass	Passed
Check run progress on DF	Send run command as specified above and do mouse over the image	The run progress status is shown below the image	Passed
Run experiment from EE page and check run progress on EE page	Press "Run" button	Run ID is updated.	Passed
		The current running executor start blinking	Passed
		"Run Result" button shows up on the finished executor	Passed
Check experiment result	Press on "Run Result" button to see the execution result	"Run Result" dialog pops up showing all the execution result	Passed

9.1.2. Non-functional Requirements Evaluation

Non-functional requirements are important to check software quality. Table 10 shows the relation between non-functional requirements and the corresponding Architectural and design decision that helped to achieve that requirement. The details of the de-

sign decision can be found in specified sections. Performance and Efficiency requirements are not the main requirements in this project; however, they will play an important role in the future improvement of this project. Therefore, the main architecture also tried to enable those requirements.

Table 10 Non-functional requirement verification

Non-functional Requirements	Design decisions which support the requirement
Modularity	Clear interface definitions (Section 6.3.2 and 6.3.3)
Modifiability	Dataflow Architectural Pattern (Section 7.3.2 8.3, and 8.5)
Usability	New GUI with jsPlumb (Section 8.2)
Functional appropriateness	Dataflow Architectural Pattern (Section 7.3.2 8.3, and 8.5)
Availability	Client-Server Architectural Pattern (Section 6.2.2)
Adaptability	Technology choices (Section 7.2.2, 7.2.3, 7.2.4, 7.3.4, and 7.3.5)
Installability	xampp, Jetty (Section 7.2.4 and 7.3.4) and Development documentations
Performance & Efficiency	Even-Driven Architectural Pattern (6.3.3)

9.2 Validation

An example specified in Section 2.5 is used to show how the EE Framework helps to perform a Design-Space Exploration for a multiprocessor system. The user provides the set of values per node and EE shows the comparison of performance results for both values. To perform the entire experiment the following steps are needed:

- Use a pre-defined POOSL model: A POOSL model is provided to the project specifying a task graph and a multiprocessor system. The output files generated by POOSL model simulation are used by the Trace to produce Design Space graphs.
  - Define an experiment and download an EE model: Upload the pre-defined POOSL model to EE. The added model will be shown in the list of models located on the right side of the pane.
  - Define the experiment by adding available executors from the left side and models from the right side and connect them.
  - Click on the Download button to download the EE model to the local disk.
- Add an EE model and execute the experiment from DF
  - Add the download EE model file to a transformation in DF.
  - Create input/output parameters and connect them to the 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 the execution can be triggered. In the DSE example, the user needs to map seven tasks to four nodes, specify the node types, set the maximum execution time units, and declare desired throughput. The user can provide multiple values per parameter by listing them with comma between. During the execution, the output image part displays an hourglass and the progress status appears below the image. After the entire execution flow completes, the DF is updated with Design Space Graphs to display the comparison result of missed deadlines and desired throughputs.

This entire process shows how DSE can be done with a single case. The looping functionality helps the user to save all the manual procedures and achieve the automated experiment. However, in real life, the model can be extremely large and complex having thousands of parameters. In that case, the execution time can be extremely big and the parameter value assignment can be a cumbersome job. Therefore, performance and usability would be a main topic for future work.

10. Conclusions

Abstract – In this chapter, the results obtained in the project are covered. Following this, the lessons learned throughout the project are given. Moreover, the suggestions for future work, which will be needed to improve the product and make it more complete, are listed.

10.1 Results

Within the scope of CoDeX, a new tool Exploration Experiment (EE) was developed for TNO-ESI. The CoDeX with automated DSE example was demonstrated as a result.

During the development, many technical investigations took place that helped to address certain problems encountered in this project. The following are the achievements in this project:

- Good problem and solution analysis
- Good architecture and design
  - A modular architecture with the help of precise interface definitions (Section 6.3.2 and 6.3.3)
  - A modifiable and functionally appropriate architecture with the help of Dataflow Architectural Pattern (Section 7.3.2, 8.3, and 8.5)
  - An available architecture with the help of Client-Server Architectural Pattern (Section 6.2.2)
  - Adaptable and Installable architecture with the help of technology choices (Section 7.2.2, 7.2.3, 7.2.4, 7.3.4, and 7.3.5).
  - Fast and efficient architecture with the help of Event-Driven Architectural Pattern (Section 6.3.3)
  - User friendly GUI with the help of intuitive design (Section 8.2)
- Demonstration of the automated DSE as a proof that the concept is technically realizable (Section 9.2)

10.2 Future works

Following is the list of the suggestions for future improvements.

- EE Definition Handler
  - Currently, a graph drawing library, jsPlumb, is used to draw graphs and the interface is implemented from scratch. Instead, a ready-to-use library discovered during the investigation, NoFlo [31], can be used for complete functionality.
- EE Execution Handler
  - Also, a ready-to-use library, JavaFBP [30], can be used for complete functionality.
  - The root path (c://dse/) of files and folders should be configurable
- DF
  - Parameter value assignment can be done in different way to improve efficiency
  - A run button is needed in DF's interface; otherwise, the execution is triggered with any small changes, which prohibits the user from making several changes at once.
- A version control mechanism is needed to track changes in the experiment definition.

- However, performance and security are not within the scope of this project. They should be main concerns of the future works.
- User authentication functionality can be added for security reasons.
- In the POOSL, observer functionality is needed to catch the output parameters in the first place; hence, DF can better handle the output results.
- EE needs to be tested with other tools and models that are used in real situations to further analyze the problem and requirements.

11. Project Management

Abstract – The previous chapters described what the project is about. This chapter focusses on how it was carried out, from a project management point of view. The focus is on the methods used to steer the planning and monitor the progress, which includes a description of work breakdown structure and project planning.

11.1 Introduction

The project was conducted for eight months and had two big iterations. The first iteration continued during the first five and half months and aimed to have a prototype which could demonstrate the DSE example (Section 2.5) with single parameter values. The first iteration was conducted together with Fangyi Shi. The second iteration lasted last three months and was dedicated to improvement of the project. The main goal to achieve during the second iteration was to demonstrate DSE example with multiple parameter values, which gives possibility to have an automated DSE.

In the early stage of the project, the management group only had some initial opinions about the project and the requirements were not so clear. Therefore, an agile development method was applied throughout the entire project, which made the requirement definition process much easier. A weekly meeting was held; hence, the completed tasks could be confirmed quickly and the next tasks were discussed and defined.

11.2 Important Dates

Figure 24 shows the important events, milestone start dates, and deadlines in the project. Most important dates at the company are a TNO-ESI Symposium [32] on September 16, 2014. Lots of potential future customers came to that event and it was crucial to have a nice demonstration as a result of the first iteration. The symposium was held successfully and received lots of useful feedback from the symposium participants.

Another important event related to the university are the Comeback Day Presentation and Final Presentation.

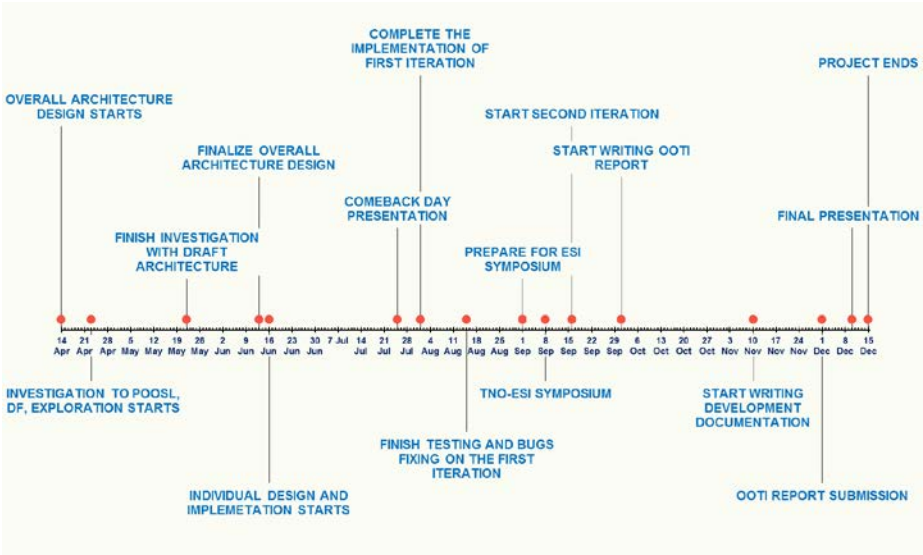

Figure 24 Important project events

11.3 Initial Project Schedule

Figure 25 shows the initial project schedule built at the beginning of the project. The Overall Architecture definition part is the heaviest part which requires lots of investigation, domain study, problem and requirement analysis and a solution for those. We planned almost two months for Overall Architecture and one and half month for Design and Implementation part. Also, we saved almost one whole month for testing and bug fixing part, though, the estimated cost was two weeks. Two weeks of buffer zone was needed in case any big change is needed based on the result of the first prototype.

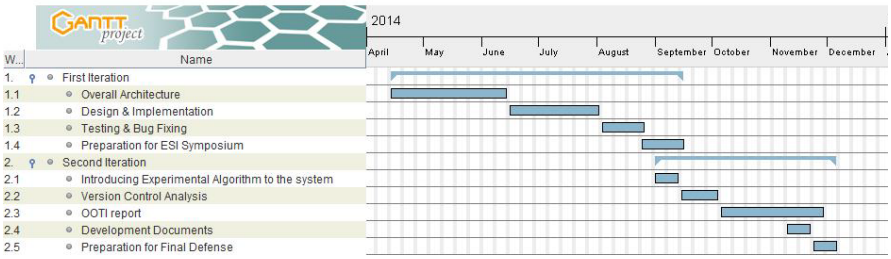

Figure 25 Initial Project Schedule

11.4 Schedule Change

Figure 26 shows the schedule changes that occurred during the project. Blue bars represent initial plan and red bars represent shifted plan. The overall project was delayed by one week, which fits within the buffer zone.

11.4.1. New Requirements

The first iteration went exactly according to the plan; however, new requirements (detailed in Sections 8.2.2 and 8.2.3) popped up as expected at the end. The customer wanted a new GUI that could have better graph drawing possibilities. The other requirements were basically related to the EE Definition Handler improvement. The

estimated time for the new GUI with graph drawing possibilities was three weeks and the rest of requirement was two weeks. My teammate Fangyi Shi was responsible for the GUI implementation; however, this requirement was not feasible for her because of the time limitations. Therefore, I took the responsibility of implementing all the new requirements because one of the main requirements, Introducing Experimental Algorithm for automated DSE, was not possible to achieve without a new GUI.

11.4.2. Investigation Difficulty

One of the requirements scheduled in the second iteration, introducing an experimental algorithm for automated DSE, was really a challenging one for me to achieve because author and other supervisors do not have enough knowledge on that and extensive research was needed to find a solution. The initial estimation for this requirement was two weeks but it took five weeks as my supervisors expected. Most of the time was spent on investigation; therefore, a little time was left for the implementation. However, we already agreed on limiting generality in view of available time. Therefore, we have decided to implement very a simple case and future improvements are left for the next developers. The nice to have requirement, version control analysis, also came up after the first iteration. This requirement also shifted to the future work list because of the time limitation.

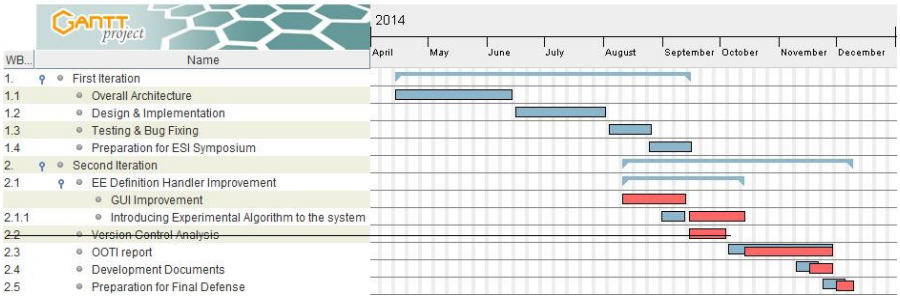

Figure 26 Schedule Change

11.5 Conclusions

The most important event, demonstration for the TNO-ESI Symposium, was held successfully. Based on the successful result, new requirements popped up and most of them were covered in iteration two. The main goal of the iteration two, Experimental Algorithm, was implemented with a simple case and proposal for the complete solution is covered. The customers are happy with the overall achievement.

12. Project Retrospective

Abstract – In the previous chapter the project was evaluated from a project management point of view. In this chapter the project is evaluated again, but this time as a personal reflection. What went as expected, what did not, and what are the lessons learned? Moreover, the design opportunities from Section 3.3 are revisited.

12.1 Reflection

When I look back at the CoDeX project after working on it for eight months, the overall project looks successful. This project was different than the other projects I experienced before because I was in charge of the entire process, this gave me the possibility to experience all different roles such as the requirement engineer, system architect, designer, technical expert, developer, tester, and user at the same time.

Also, this project gave me the full experience of Software Engineering in real practice. I could apply the theoretical knowledge gained during the OOTI education in real life, which made my knowledge more concrete.

The quote, *Design is taking decisions such that the happiness increases* by dr.Kees van Overveld, always stayed in my mind during the project and I aimed to make the customers happy with the design decisions I made.

In this project, the design decisions were made based on intensive research and technology investigation. As a result, I could achieve a good architecture, which made the customers happy. The most joyful moments in the project were seeing a happy customer after long hard work.

Also, I could grow a lot as a Software Architect and Designer with help of the following lessons.

12.1.1. Lessons Learned

Lessons learned from the technical point of view:

- Architectural patterns: In this project we applied several architectural patterns which helped us to solve certain problems. Finding that patterns was not that easy, requiring several weeks of reading. As a result, now I have a better understanding of Architectural Patterns and their relations with the quality attributes. I am motivated to learn more about Software Architecture and Design and am looking forward using them in the future projects.
- Also, with a help of the project, I could earn solid knowledge on websocket, polling, and webservice, the difference between them, and when to use them.
- The project helped me to understand the extremely interesting domain, DSE, better.

Lessons learned from an organizational point of view:

- I realized that I should decrease the time of thinking alone and increase the time of discussing with people. Of course, it does not make sense if I just try to talk with others while I know really nothing. Therefore, finding a balance is important.

- It is impossible to accomplish everything on the table in limited amount of time. I should prioritize all the requested items and if certain task is not feasible, I should be able to decline the request and put it in the future work list.

12.2 Design opportunities revisited

In Section 3.3, assessment criteria for a technical design that are very relevant for this project were identified. Below, for each of these criteria, it is verified whether the design adheres to that criterion.

- The artifact satisfies all the requirements described in Section 5 that are validated and verified in Section 9.
- The system is *easy to use* because the new GUI designed and implemented second prototype (Section 8.2) qualifies all usability requirements. Also, the system is easily adaptable and installable with help of the design and technology choices (Section 7.2.2, 7.2.3, 7.2.4, 7.3.4, and 7.3.5).
- The architecture is modifiable and modular with the help of the interface definitions described in Section 6.3.2 and 6.3.3. The architecture is modifiable with help of the Dataflow architectural pattern (Section 7.3.2, 8.3, and 8.5). Therefore, the architecture is concluded as structured.
- The result of the project proves that the concept, Coherent tool support for DSE, works with the real demonstrator. Therefore, the architecture and design is convincing (Section 9.2).
- The real demonstration for automated DSE shows that the concept is technically realizable (Section 9.2).
- The presentation of the artifact is elegant with the implementation of intuitive GUI (Section 8.2).

Glossary

4+1 Architecture view model	It organizes a description of a software architecture using five co current views, each of which addresses a specific set of concerns.
Activity Diagram	UML graphical representation of workflows of stepwise activities and actions
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.
CoDeX	Coherent Tool Support for Design-Space Exploration
Component Diagram	UML graphical representation that depicts how components are wired together to form larger components and or software systems
Database	An information organized in such a way that a computer program can quickly select pieces of data
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.
EDA	Event-Driven Architecture
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.
Experiment	An experiment contains a sequence of a model and executors.
FBP	Flow Based Programming
GUI	Graphical User Interface
HTML5	HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web.
HTTP	The Hypertext Transfer Protocol. It is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web

IDE	An integrated development environment is a software applica-tion that provides comprehensive facilities to computer pro-grammers for software development.
Java Archive	JAR (Java Archive) is a package file to distribute application soft-ware 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 con-tainer 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.
Long polling	
MVC	Model-View-Controller design pattern.
MySQL	It is a widely used open-source relational database management system.
OOTI	Onwerpersopleiding Technische Informatica
OpenGL	Open Graphics Library is a cross-language, multi-platform appli-cation programming interface (API) for rendering 2D and 3D vec-tor graphics.
PHP	PHP is a server-side scripting language designed for web devel-opment but also used as a general-purpose programming language.
Polling	
POOSL	It provides an integrated editing, debugging and validating envi-ronment for POOSL modelling, combined with a high-speed simu-lator.
Rotalumis	It is the high-speed simulator for POOSL models.
SOA	Service-Oriented Architecture
ST	Software Technology
TCP	The Transmission Control Protocol (TCP) is one of the core proto-cols of the Internet protocol suite (IP), and is so common that the entire suite is often called TCP/IP
TNO-ESI	Embedded Systems Innovation by TNO
TRACE	It is a tool for visualizing quantitative analysis results.
TU/e	Eindhoven University of Technology
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.
XAMPP	Cross platform web development environment which allows de-velopers to create web applications with Apache2, PHP and a MySQL database.

Bibliography

[1] About TNO-ESI, http://www.esi.nl/about-tno-esi/
[2] About TNO-ESI Tools, http://www.esi.nl/solutions/
[3] Twan Basten, Emiel van Benthum, and partners, Model-Driven Design-Space Exploration for Embedded Systems: The Octopus Toolset, Embedded Systems Institute, Eindhoven University of Technology, Radboud University Nijmegen, Oc´e Technologies
[4] About Design Framework, http://df.esi.nl/
[5] About Formulas in Design Framework, http://df.esi.nl/howtouse2/models.dot
[6] TNO-ESI POOSL, http://poosl.esi.nl/.
[7] TU/e POOSL, http://www.es.ele.tue.nl/poosl/.
[8] TNO-ESI TRACE, http://trace.esi.nl/.
[9] 5kk80 Assignment 2 – Design-Space Exploration, Eindhoven University of Technology, Department of Electrical Engineering, April 2008
[10] Ramon R.H. Schiffelers, Wilbert Alberts, Jeroen P.M. Voeten, Model-based specification, analysis and synthesis of servo controllers for lithoscanners, 2012
[11] Kees van Hee and Kees van Overveld, New criteria for assessing a technological design, April 2012
[12] ISO SQuaRe Product Quality, http://maisqual.squoring.com/wiki/index.php/ISO/IEC_SQuaRE
[13] Fangyi Shi, Coherent Tool Support for Design-Space Exploration, October 2014
[14] Paul Clements and Len Bass, Software Architecture in Practice, 2012
[15] Event-Driven messaging, http://soapatterns.org/design_patterns/event_driven_messaging
[16] Websocket vs polling, http://www.websocket.org/quantum.html
[17] MINI, http://www.php-mini.com/
[18] PHP Coding Style, http://www.php-fig.org/psr/psr-2/
[19] PHP Data Objects, http://php.net/manual/en/intro.pdo.php
[20] CakePHP, http://cakephp.org/
[21] XAMPP, https://www.apachefriends.org/index.html
[22] Johnston, Wesley M.; J.R. Paul Hanna, Richard J. Millar, Advances in Dataflow Programming Languages, ACM Computing Surveys, March 2014
[23] Wadge, William W.; Edward A. Ashcroft, Lucid, the Dataflow Programming Language, Academia Press, August 2013
[24] Jetty WebSocket Server API, http://www.eclipse.org/jetty/documentation/9.1.5.v20140505/jetty-websocket-server-api.html
[25] jsPlumb, http://www.jsplumb.org/demo/flowchart/dom.html
[26] JointJS, http://jointjs.com/
[27] Diagramo, http://diagramo.com/
[28] GoJS, http://gojs.net/
[29] Flow-Based Programming, http://www.jpaulmorrison.com/fbp/
[30] JavaFBP, http://www.jpaulmorrison.com/fbp/software.html#JavaFBP
[31] NoFlo, http://noflojs.org/
[32] TNO-ESI Symposium, http://www.esi.nl/innovation-support/sharing-know-how/TNO-ESI-Symposium/

About the Authors

Bayasgalan Baatar received her BSc degree from National University of Mongolia in 2007. In her BSc thesis she designed and developed a Customer Relationship Management System for the Investor Education Organization. After her graduation she worked at National University of Mongolia as an Engineer of Information Systems for one and a half year. She received her MSc degree from University of Tsukuba in 2012. For her MSc thesis she made a comparison between sensor based and vision based techniques for dynamic gesture recognition. During her internship at Fujitsu Co.,LTD she worked on Cloud Computing System development. Also, she won a Tsuji Asia scholarship from 2011 to 2012. Bayasgalan has been enrolled in the PDEng program at TU/e since 2012, designing software for industry and research.

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.