Design Framework Process Tailoring

Martin Palatnik
September 2012
Design Framework Process
Tailoring
Martin Palatnik
September 2012
Design Framework Process Tailoring

Martin Palatnik

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

Embedded Systems Institute  
Eindhoven University of Technology

Steering Group

Roelof Hamberg
Teade Punter
Peter Vink
Gerard Zwaan

Date

September 2012
Abstract
The development of high-tech electromechanical systems is complex and requires a significant effort. In such environments it is evidently difficult to keep the key design information and rationales synchronized. The Design Framework is a visual modeling tool that aims to help architects and designers to capture this information. The professional high-speed printers design is an ideal scenario where the Design Framework can be used. However, the size of the projects, the flexibility in creating views of the information, and the seamless integration with analysis models are some challenges to use it in such a scenario. The Design Framework Process Tailoring is an adaptation of the framework to be used in the design process of professional high-speed printers. This tailoring is the consequence of a thorough analysis of the original concept and of the professional high-speed printers design process. The result is a set of extensions that allow the architects and designers to use the Design Framework in larger and more complex projects, connect to specific company models, and provide more powerful visualizations of the information.

Keywords
systems engineering, model driven design, visual modeling tool, design process, multidisciplinary design, information modeling, rationale tracking

Preferred reference

Partnership
This project was supported by Eindhoven University of Technology and Embedded Systems Institute.

Disclaimer Endorsement
Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology or Embedded Systems Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or Embedded Systems Institute, and shall not be used for advertising or product endorsement purposes.

Disclaimer Liability
While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.

Trademarks
Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Copyright
Copyright © 2012. Eindhoven University of Technology. All rights reserved.
No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and Embedded Systems Institute.
Foreword

The Embedded Systems Institute’s mission is to advance innovation and excellence in systems engineering for high tech embedded systems. An important contribution to this mission is the integration of model-based methods into multi-disciplinary systems architecting, not only by working methods described in articles, but also by usable tools that support the architects in their daily work. As these tools are not readily available, ESI started prototyping the so-called design framework over the last years, while simultaneously validating its concept with several industrial partners through workshops and live cases, raising serious attention in industry, most notably Océ, to adopt the proposed method and tool in due time.

Martin’s assignment was to make this tool fit for Océ, i.e., customize the Design Framework in such a way that architects at Océ can use it within their domain and within their processes. The initial direction that we suggested was towards adding automatic view generation and tool integration into the current prototype, the latter being developed in the Eclipse modeling framework with extensive usage of GMF as well. In short, the network of different stakeholders, the exact functionality of the tool, as well as the tool’s realization, all of these factors were quite entangled and non-trivial. Therefore, challenges were to be found in most, if not all, of the views of the CAFCR architecting method: Customer, Application, Functional, Conceptual, Realization.

This challenge fitted Martin very well. It fitted who he is as a person, with lots of energy, enthusiasm and open-mindedness to listen to problems. He also had the drive to focus and solve these problems. Furthermore, Martin cooperated with us in way of providing direct feedback, orally as well as with gesturing when he was (dis)liking the discussions. It matched what Martin does and his pro-activity to get things going. It fitted his technical competences at different levels: understanding concepts of usage, concepts of realization, the implementation itself, and the Eclipse framework. This all resulted in a very significant step in the development of the Design Framework. It is now possible to work with a large project team in different flows, separately and together at the same time. Users can integrate model information in a bi-directional way: from the Design Framework to the model and vice versa. And last but not least, Martin has added a number of improvements with a very positive effect on the stability of the Design Framework. As a whole, the development step that was taken has indisputably enabled the important industry-as-laboratory experiments that have to follow in the near future.

Martin, we thank you for this contribution to the Design Framework and wish you a lot of success in your future career.

Eindhoven, September, 2012

Roelof Hamberg, Peter Vink, Teade Punter
Embedded Systems Institute
Preface

This technical report presents the results of the Design Framework Process Tailoring project. This project was carried out as a final assignment for the post Master’s Software Technology program at Eindhoven University of Technology as a part of the larger Octo+ project. Octo+ is a joint project between the Embedded Systems Institute (ESI) and Océ-Technologies B.V (Océ) to streamline the use of models in the design process.

In this report the design and development of a set of extensions for the Design Framework to better represent the design process of high-tech printers is presented. For readers not familiar with the Design Framework and the design process of high-tech printers Chapter 3 is a good beginning. For readers interested in the design of the solution Chapters 4 and 5 are the most relevant. For readers only interested in the goals and results of the project Chapters 1 and 8 provide a sufficient overview.

Martin Palatnik
September 6, 2012
Acknowledgements

A project is never done in isolation. This project is not an exception to this rule and without the help and guidance of several people it could never have been done.

First, I would like to thank my supervisors, Roelof Hamberg, Teade Punter, Peter Vink and Gerard Zwaan for their guidance, feedback and discussions during the entire project. From their different perspectives they helped this project to be successful and they taught me several good lessons in between.

I would like also to thanks Frans Reckers who provided many useful tips and Hristina Moneva who was my supervisor during the first months.

This project would not have been possible without Océ-Technologies B.V. I would like to thank all the architects, designers and engineers who I met in the different stages of the project. Particularly I would like to thank Guy Stoot for making himself available every week to have interesting discussions and Patrick Vestjens for his valuable feedback at the beginning of this project.

I would like to express my special gratitude to Ad Aerts for his valuable input and guidance during the two years of OOTI and Maggy de Wert for her support during my time in the Netherlands.

I would also like to express a very special gratitude to all my OOTI colleagues for making the past two years an incredible experience. Especially to my good friends Alex Loizidis, Fanis Grollios, Diana Ahogado, Ivana Kostadinovska, Tudor Mihordea, Francisco Heredia and Ariel Vargas who were my family in the Netherlands.

Last but not least, I would like to thank my parents Elena and Jorge for their unconditional support, my life friends in Uruguay for supporting me regardless of the distance and my love Estela who is always there.

Thanks.

Martin Palatnik
September 6, 2012
Executive Summary

The Design Framework is a visual modeling tool that aims to help architects and designers to develop complex systems. The Design Framework accomplishes this mission by capturing the design rationales in the design process and by providing a mechanism for using related to heterogeneous models.

However, the Design Framework needs some adaptations to be used in the design process of high-speed production printers. The Design Framework has some difficult challenges to:

- Represent very large projects that involve many multidisciplinary teams;
- Create flexible views of the information;
- Integrate seamlessly with analysis models.

This report describes the result of a project to design and implement a set of extensions to the Design Framework to address and solve these issues. These extensions are installed on top of the original Design Framework without modifying the core functionalities. These extensions were tested in some company scenarios to validate its functionality.

The results of this project present an important step towards the validation of the Design Framework research and concept and its adoption in an industrial context.
# Table of Contents

Foreword .................................................................................................................. i

Preface ....................................................................................................................... iii

Acknowledgements ................................................................................................... v

Executive Summary ................................................................................................. vii

Table of Contents ...................................................................................................... ix

List of Figures ............................................................................................................ xiii

List of Tables .............................................................................................................. xv

1 Introduction ........................................................................................................... 1
   1.1 Context ............................................................................................................. 1
   1.2 Organizations ................................................................................................. 1
   1.3 Design Framework Process Tailoring ............................................................. 2
   1.4 Outline ............................................................................................................. 2

2 Stakeholders .......................................................................................................... 4

3 Domain and Problem Analysis ............................................................................ 6
   3.1 Domain Analysis ............................................................................................. 6
       3.1.1 The Design Framework ........................................................................... 6
   3.2 Problem Analysis ............................................................................................ 8
       3.2.1 Problem Statement .................................................................................. 8
       3.2.2 Analysis .................................................................................................. 8
   3.3 Project Goal .................................................................................................... 13

4 System Requirements ........................................................................................... 14
   4.1 Scope ................................................................................................................ 14
   4.2 Functional Requirements .............................................................................. 14
       4.2.1 Project, phases and multiple flows ....................................................... 14
       4.2.2 Model integration .................................................................................. 15
       4.2.3 Search .................................................................................................... 15
       4.2.4 Relationship exploration ...................................................................... 16
   4.3 Non-Functional Requirements ...................................................................... 16
       4.3.1 Model integration Non-Functional requirements ............................... 16
   4.4 Constraints ..................................................................................................... 16
   4.5 Use cases: Project, phases and multiple flows ............................................ 17
       4.5.1 Create a project diagram ...................................................................... 17
       4.5.2 Add a new phase .................................................................................... 17
       4.5.3 Add a flow .............................................................................................. 18
       4.5.4 Open a flow from the Phase editor ....................................................... 19
4.5.5 Add a flow dependency ................................................................. 19

4.6 Use Cases: Model integration ......................................................... 20
4.6.1 Attach an external model ......................................................... 20
4.6.2 Run an external model ............................................................ 21
4.6.3 Import information from an external model ............................... 22
4.6.4 Export information to an external model ................................. 22
4.6.5 Register a model provider ....................................................... 23

4.7 Use Cases: Search .................................................................... 23
4.7.1 Search for an element ............................................................. 23
4.7.2 Select an element resulting from the search .............................. 24

4.8 Use Cases: Relationship exploration ........................................... 24
4.8.1 Open the relationship exploration .......................................... 24
4.8.2 Explore .............................................................................. 24

5 System Architecture and Design ................................................. 26

5.1 Introduction ............................................................................ 26

5.2 Main design decisions ............................................................. 27
5.2.1 Design Decision: Expand the DF ........................................... 27
5.2.2 Design Decision: Extendable model integration .................... 28

5.3 Architecture overview ............................................................ 29

5.4 Logical View .......................................................................... 30
5.4.1 DF Core meta-model ............................................................ 30
5.4.2 DF Octo+ meta-model .......................................................... 31
5.4.3 Editors .............................................................................. 32
5.4.4 The DF Core editors ........................................................... 33
5.4.5 The DF Octo+ editors ......................................................... 36
5.4.6 Model integration ............................................................... 36

5.5 Development view ................................................................. 37
5.5.1 EMF meta-model generated classes ..................................... 39
5.5.2 DF meta-model generated classes ....................................... 40
5.5.3 DF Octo+ meta-model generated classes .............................. 40
5.5.4 GMF editors .................................................................... 41
5.5.5 The DF editors generated classes ....................................... 43
5.5.6 The DF Octo+ extension editors generated classes .............. 43
5.5.7 The Flow and View editors extensibility ............................... 43
5.5.8 Model integration extensibility .......................................... 43
5.5.9 Model integration user interface independence ..................... 47

5.6 Deployment View .................................................................... 48
5.6.1 Eclipse plug-ins ................................................................. 48
5.6.2 Typical configuration ........................................................... 49

5.7 Process View .......................................................................... 50
5.7.1 Main processes ................................................................ 50

6 Verification & Validation ............................................................. 52

6.1 Verification ............................................................................ 52
6.1.1 Use cases verification ......................................................... 52
6.1.2 Additional test cases ........................................................... 52

6.2 Validation .............................................................................. 56
6.2.1 Scenario 1: Project, phases and multiple flows modeling of one of the current printer design projects ........................................ 56
6.2.2 Scenario 2: Model integration of some company specific models ................................................................. 56

7  Project Management ................................................................. 57
   7.1 Way of working ................................................................... 57
   7.2 Project Planning and Scheduling ......................................... 57
   7.3 Risks ............................................................................... 58
   7.4 Project retrospective .......................................................... 59
   7.4.1 Strengths ..................................................................... 59
   7.4.2 Improvements ............................................................... 59

8  Conclusions ............................................................................ 60
   8.1 Results ............................................................................. 60
      8.1.1 Project, phases and multiple flows .............................. 60
      8.1.2 Model integration ....................................................... 63
      8.1.3 Relationship exploration ............................................ 64
      8.2 Reflection .................................................................... 64
      8.2.1 Applicability of the solution ....................................... 65
      8.2.2 DF Meta-model extensibility layer .............................. 65
      8.2.3 DF Meta-model .......................................................... 66
      8.2.4 EMF and GMF Model Driven Development ................ 66
      8.3 Future Work .................................................................. 67

Glossary ..................................................................................... 69

Bibliography ............................................................................... 71

About the Author ........................................................................ 73
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stakeholders in the printer manufacturing company, ESI and TU/e</td>
</tr>
<tr>
<td>2</td>
<td>Design decision and system design snapshot (from D5.1.2 [5])</td>
</tr>
<tr>
<td>3</td>
<td>System block (from D5.1.2 [5])</td>
</tr>
<tr>
<td>4</td>
<td>Models (from D5.1.2 [5])</td>
</tr>
<tr>
<td>5</td>
<td>Multiple flows</td>
</tr>
<tr>
<td>6</td>
<td>Design phases of a project</td>
</tr>
<tr>
<td>7</td>
<td>Synchronization of the design between flows</td>
</tr>
<tr>
<td>8</td>
<td>Model Provider Repository</td>
</tr>
<tr>
<td>9</td>
<td>New project diagram created and opened</td>
</tr>
<tr>
<td>10</td>
<td>Project diagram with added phases</td>
</tr>
<tr>
<td>11</td>
<td>Multiple flows in the Phase editor</td>
</tr>
<tr>
<td>12</td>
<td>Visualization of the flow dependency</td>
</tr>
<tr>
<td>13</td>
<td>Attach a model</td>
</tr>
<tr>
<td>14</td>
<td>Run the model</td>
</tr>
<tr>
<td>15</td>
<td>Search option</td>
</tr>
<tr>
<td>16</td>
<td>Relationship exploration</td>
</tr>
<tr>
<td>17</td>
<td>DF Core component extension</td>
</tr>
<tr>
<td>18</td>
<td>Model integration</td>
</tr>
<tr>
<td>19</td>
<td>Architecture overview</td>
</tr>
<tr>
<td>20</td>
<td>DF meta-model part 1</td>
</tr>
<tr>
<td>21</td>
<td>DF meta-model part 2</td>
</tr>
<tr>
<td>22</td>
<td>Project, phases and multiple flows meta-model</td>
</tr>
<tr>
<td>23</td>
<td>Model integration meta-model</td>
</tr>
<tr>
<td>24</td>
<td>Editor conceptual model</td>
</tr>
<tr>
<td>25</td>
<td>Model integration conceptual structure</td>
</tr>
<tr>
<td>26</td>
<td>Model integration dynamic structure</td>
</tr>
<tr>
<td>27</td>
<td>EMF generated packages decomposition</td>
</tr>
<tr>
<td>28</td>
<td>DF meta-model package decomposition</td>
</tr>
<tr>
<td>29</td>
<td>DF Octo+ meta-model package decomposition</td>
</tr>
<tr>
<td>30</td>
<td>Meta-model packages dependencies</td>
</tr>
<tr>
<td>31</td>
<td>GMF generated packages decomposition</td>
</tr>
<tr>
<td>32</td>
<td>Parts, Policies and Commands Structure</td>
</tr>
<tr>
<td>33</td>
<td>Add Edit Policies</td>
</tr>
<tr>
<td>34</td>
<td>Design decision creation part 1</td>
</tr>
<tr>
<td>35</td>
<td>Design decision creation part 2</td>
</tr>
<tr>
<td>36</td>
<td>Plug-in architecture structure</td>
</tr>
<tr>
<td>37</td>
<td>Plug-in architecture sequence</td>
</tr>
<tr>
<td>38</td>
<td>Model integration structure</td>
</tr>
<tr>
<td>39</td>
<td>DF Eclipse plug-ins and features</td>
</tr>
<tr>
<td>40</td>
<td>Typical configuration</td>
</tr>
<tr>
<td>41</td>
<td>Eclipse process</td>
</tr>
<tr>
<td>42</td>
<td>Command line process</td>
</tr>
<tr>
<td>43</td>
<td>High level schedule</td>
</tr>
<tr>
<td>44</td>
<td>Iteration effort estimation and tracking</td>
</tr>
<tr>
<td>45</td>
<td>Project editor</td>
</tr>
<tr>
<td>46</td>
<td>Phase editor</td>
</tr>
<tr>
<td>47</td>
<td>Extended view Editor + Phase Editor</td>
</tr>
<tr>
<td>48</td>
<td>Extended Flow editor</td>
</tr>
<tr>
<td>49</td>
<td>Migration of dependencies</td>
</tr>
<tr>
<td>50</td>
<td>Attaching model using provider</td>
</tr>
<tr>
<td>51</td>
<td>Map external model to DF</td>
</tr>
<tr>
<td>52</td>
<td>Parameters copied and mapped in DF</td>
</tr>
<tr>
<td>53</td>
<td>Command line model integration</td>
</tr>
<tr>
<td>54</td>
<td>Relationship exploration</td>
</tr>
<tr>
<td>55</td>
<td>Extensibility layer</td>
</tr>
<tr>
<td>56</td>
<td>Top abstractions in UML meta-model</td>
</tr>
</tbody>
</table>
List of Tables

Table 1: Roles and responsibilities of Stakeholders in ESI and TU/e ..................5
Table 2: Roles and responsibilities of Stakeholders in the printer manufacturing company .......................................................... 5
Table 3: Architecture model - DF mapping ...........................................12
Table 4: DF meta-model elements part 1 .............................................31
Table 5: DF meta-model elements part 2 .............................................32
Table 6: Project, phases and multiple flows meta-model elements ...........33
Table 7: Model integration meta-model elements ................................35
Table 8: Main risks ...........................................................................58
Table 9: Mitigation strategy ...............................................................59
1 Introduction

This report describes the Design Framework Process Tailoring project. This chapter provides the background information related to this project, briefly discusses its goal, and gives an outline of this report.

1.1 Context

The Design Framework Process Tailoring is a joint project between the Embedded Systems Institute (ESI) and Océ-Technologies B.V (Océ) that is carried out as a final assignment of the Software Technology designer program at the Eindhoven University of Technology (TU/e). This project is based on the result of a previous FP7 project called Multiform in which ESI was involved.

The Multiform project was a European research project, executed between 2008 and 2012, that focused on the development, the integration, and the interoperation of techniques and tools to provide coherent tool support for the integrated control design of large and complex networked systems [1]. ESI was responsible for one of the work-packages of that project. The result of that work-package is a tool called Design Framework (DF).

The DF is a visual modeling tool that aims to help architects and designers to develop complex systems. The DF accomplishes this mission by capturing the design rationales in the design process and by providing a mechanism for using heterogeneous models. The DF was made generic so that it can be used in different companies, and can be tailored for a company specific design process.

Océ is interested in using the DF and as a consequence sponsored the domain tailoring as a part of the larger Octo+ project. With Octo+ Océ, in a collaboration with ESI, attempts to streamline the development process by using a federation of interconnected models and by capturing the design knowledge throughout all development cycles. The long-term vision of the Octo+ project is to establish a complete virtual product development process, supported by software tooling, which eliminates the need to build physical prototypes [2].

1.2 Organizations

The main organization behind this project is ESI, a research institute that intends to advance industrial innovation and academic excellence in the embedded systems engineering field. ESI research projects are one of the key activities of the organization. There are two kinds of research projects: exploratory and applied. The exploratory research projects investigate some of the major long-term research challenges presented by embedded systems engineering in the areas of performance, reliability, security, and adaptability. The applied research projects mainly focus on the application domains of professional systems, high-volume products, high integrity and safety-critical embedded systems, and systems-of-systems applications [3]. Multiform was one of the exploratory research projects and Octo+ is one of the applied research projects.

All research projects are carried with an industrial partner. In this project, and in the context of Octo+, the industrial partner is Océ. Océ is an international company that manufactures and sells production printing and copying hardware, and related software. Océ specializes in durable, high-end equipment, suitable for corporate publishing/reproduction centers as well as commercial printing and copying operations. For the document-printing business, most equipment produced by Océ is high-speed (50
pages per minute and over) and has very high duty cycles (half a million pages per month and higher).

### 1.3 Design Framework Process Tailoring

High-speed production printers are complex products that require a significant design effort. The size of the effort can be observed through the number of people involved in the design and the number of years needed in the product development. In a typical new printer development project there can be up to 150 specialists working on the design during the mid-stages of the project and the total project length is in general several years.

In such complex projects it is very challenging to capture a consistent picture of the design. The design is done in parallel by many teams. Each team designs a specific part of the printer and in order to reduce development costs and to speed up the iterations many models are built during the process. A number of models are monodisciplinary while others are multidisciplinary. This heterogeneity of teams, disciplines, and models makes it difficult to capture a consistent picture of the design at a specific moment in time.

In this report, the DF tailoring for the design of high-speed production printers is introduced as a research solution to the aforementioned problem. The main outcome of this tailoring is a set of extensions to the DF prototype that are designed to increase the functionality of it towards the printer manufacturing company needs.

The target functionality of the tailored DF was determined through a thorough analysis of the Design Framework concept and of the production printers design process. The analysis includes the three layers of the DF: the flow layer, the view layer and the model layer, and the printers’ design process and models. This analysis and the resulting solution are described in the rest of this report.

### 1.4 Outline

This report is organized in the following chapters.

- Chapter 1 describes the context of the project, introduces the problem and provides and outline of the report.
- Chapter 2 presents the main stakeholders of the project, their role in the organizations and their responsibility in the project.
- Chapter 3 introduces the domain and problem analysis. In the domain analysis, the DF conceptual model and prototype are outlined, and in the problem analysis they are contrasted with the printers’ design process to discover the main adaptations required.
- Chapter 4 gives a detailed overview of the systems requirements in terms of functional requirements, non-functional requirements and use cases. The requirements are based on the problem analysis of Chapter 3.
- Chapter 5 describes the architecture and design of the solution. This chapter uses the 4+1 architectural model and presents: a logical view, a development view, a deployment view, and a process view where the main artifacts and design choices are detailed.
- Chapter 6 discusses the verification and validation methods and results. The primary focus of this chapter is to answer the following questions: Has the product been built right? Has the right product been built?
• Chapter 7 introduces an overview of the project management process. It includes a description of the way of working, the planning, the risks and a retrospective of the planning and execution of the project.

• Chapter 8 presents the results of the project and the future work.
Several stakeholder roles were identified in the three organizations involved. Figure 1 shows the different stakeholder roles. The arrow indicates that a stakeholder, who is in a position between several organizations, is behind the interests of the organization that is pointed to. The black heads indicate the main stakeholders.

Figure 1: Stakeholders in the printer manufacturing company, ESI and TU/e
Table 1 and Table 2 list the responsibilities of the stakeholders and their relationship to the Design Framework. The main stakeholders are marked with a *.

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibility</th>
<th>Relationship to the Design Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Project Manager between ESI and printer manufacturing company</td>
<td>Manages the joint project and models part of a case study of the application of the Design Framework.</td>
<td></td>
</tr>
<tr>
<td>*Designer of the Domain Tailoring (Between ESI, the printer manufacturing company and TU/e)</td>
<td>Analysis of the requirements for the adaptation of the Design Framework for the printer manufacturing company. Design and implementation of the Requirements.</td>
<td></td>
</tr>
<tr>
<td>*Supervisor</td>
<td>Main developer of the Design Framework. Supervises the Tailoring of the Design Framework for the printer manufacturing company mostly from the technology perspective.</td>
<td></td>
</tr>
<tr>
<td>*Project Manager</td>
<td>Manages the Design Framework project. Supervises the Domain Tailoring from a management perspective.</td>
<td></td>
</tr>
<tr>
<td>*Supervisor</td>
<td>Supervision of the Design Framework Process Tailoring project from TU/e perspective. Involved mainly in the quality of final report.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Roles and responsibilities of Stakeholders in ESI and TU/e

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibility</th>
<th>Relationship to the Design Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architect</td>
<td>Responsible for some non-functional requirement of the printer development</td>
<td>Takes high level decisions by negotiating with different people in different roles. He makes high level models and experiments and enters requirements information in the Design Framework.</td>
</tr>
<tr>
<td>Subsystem Responsible</td>
<td>Responsible for the design of a subsystem</td>
<td>Makes design decisions for a subsystem. Makes and uses models and experiments.</td>
</tr>
<tr>
<td>Designer</td>
<td>Designs a subsystem</td>
<td>Makes models and experiments in a subsystem. Influences the decisions.</td>
</tr>
<tr>
<td>Engineer</td>
<td>Implements components in a subsystem</td>
<td>Implements components based on the designs and design decisions. Makes and uses detailed models and experiments.</td>
</tr>
</tbody>
</table>

Table 2: Roles and responsibilities of Stakeholders in the printer manufacturing company
3 Domain and Problem Analysis

Abstract – This chapter provides a detailed analysis of the domain and the problem. It explains the DF conceptual model and the prototype implemented according to this model. This chapter also presents the project goal.

3.1 Domain Analysis

3.1.1 The Design Framework
The DF is a visual modeling tool that aims to help the system architects and designers to describe a system design and reason about it by:
- Storing design decisions and their rationale.
- Providing multi-disciplinary system overview through multiple views.
- Storing key design parameters of the design.
- Storing the models and experiments done during the design process.

The DF is based on a conceptual model that defines the elements needed to describe a system design [4]. This conceptual model is a result of an analysis of the systems engineering and model-based design fields. The main line of reasoning behind it is that both the design process/activity and the design result/status are important to describe and understand a system design. As a consequence, elements that represent both concepts need to be part of it. The conceptual model is decomposed into three layers: design flow, design views and models.

The design flow represents the design process of the architects and designers. This process consists of evaluating different design decisions and choosing between them to iteratively refine the system design. In each design decision, the architects and designers have a goal and a number of design questions that are answered or hypothesized by making models, analyzing them and determining design parameters.

This process can be visually represented by a tree-like structure, as shown in Figure 2. The nodes are the system design snapshots at a certain time, and the edges are the design decisions that led to those system designs snapshots.

Figure 2: Design decision and system design snapshot (from D5.1.2 [5])

The models and parameters defined in each design decision can describe different aspects of the system design.

The design views represent the different aspects of the system design. For instance, a design view can show a discipline-based decomposition, e.g., electronics, mechanics or a system aspect based decomposition, e.g., performance or safety. A design view contains a specific decomposition of the system design that is represented by the so-called system blocks. The blocks serve as containers for the models, the parameters and the relations between them, as shown in Figure 3.
The models describe a part of the system in a specific formalism and are used to perform different analyses. From the point-of-view of the DF, the models are black boxes that need input information and produce output information. The DF does not understand the model formalisms by itself, but it can connect to these models by the key input and output design parameters. These parameters can in turn be connected between themselves with the parameter dependencies allowing the interoperability of the models. The models also have assumption parameters, which are model constants, used to explicitly store the assumptions made in the model. Model analysis runs can be executed by tools with certain tools parameters. An overview is shown in Figure 4.

The DF conceptual model introduced was used as a starting point to develop a prototype. The prototype is developed with the assistance of the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF). The EMF is used to describe the meta-model that formally defines the conceptual model previously explained and the GMF is used to define the editors that are used to create and edit information in the DF.

There are two editors to manipulate the meta-model in the DF prototype: the Flow editor and the View editor. The Flow editor represents the Flow layer and holds the
elements defined in this layer. The View editor represents the View layer and holds the elements defined in the view and model layers.

The DF prototype is deployed as a set of plug-ins to the Eclipse IDE. These plug-ins can be downloaded from the ESI multiform site df.esi.nl and installed easily in Eclipse, provided that the required versions of some other plug-ins are present in the Eclipse installation.

The prototype was designed to be extensible. Other plug-ins can extend the meta-model and provide new editors or modify the existing ones. A more detailed explanation of the DF prototype design can be found in the System Architecture and Design chapter and in the DF Multiform report [5].

### 3.2 Problem Analysis

#### 3.2.1 Problem Statement

In the past years the printer manufacturing company involved in this project has realized an important effort to optimize the design process. As a part of this effort, in collaboration with ESI, it has boosted the usage of models in the design process. The models allow the company to reduce the overall development time by creating fewer physical prototypes. For example, multidisciplinary models such as Happy Flow [6] for the paper path design, DPML [7] for the data path design and MoBasE [8] for key design parameters for the embedded software were developed during this period.

An open issue is to connect all the models and the design information produced throughout the phases of the design process. Several of these models remain independent and are used only by some individual teams. Hence, there is not a clear overview of the general design, design information might be duplicated and consistency problems can occur. In addition, the rationale behind the design is not captured and is generally lost when the project finishes or the people involved leave the company.

As one of the primary options to solve these challenges, the DF got serious attention of the printer designers who want to use it for their system designs. However, as it was mentioned in the previous sections, the DF is still a prototype based on a generic conceptual model and needs to be tailored to be used in the printer design process. On first confrontation, the printer designers perceived the size of their projects, the flexibility in creating views, and the seamless integration with analysis models as the main challenges to fit the DF in the printer design process.

#### 3.2.2 Analysis

In order to use the DF in the printer design process it is necessary to establish to what extent the DF conceptual model and prototype need to be adapted. This can be determined by analyzing the three layers of the DF conceptual model, the DF prototype and answering the following questions:

- Are the three layers sufficient to describe the design?
- Is any change necessary in the flow layer?
- Is any change necessary in the view layer?
- Is any change necessary in the model layer?
- Is the prototype usable for a production printer designer?

**Are the three layers sufficient to describe the design?**

High-speed production printers are large and complex systems composed of several sub-systems and aspects. Each sub-system or aspect is covered by different persons
or teams, led by key designers and architects. The key designers and architects are responsible for the evaluation of the design alternatives and the taking of the main decisions that affect that subsystem or aspect. Generally, these decisions are based on models and experiments made by the designer team. The accumulation of these decisions forms the aspect or sub-system design flow and represents a part of the total system design evolution.

Therefore, de facto there are multiple design flows in a printer design project, as is shown in Figure 5. The flow layer is well suited to describe one of these design flows. However, it lacks the expressivity necessary to describe multiple design flows. It is necessary to find a way to represent these multiple concurrent design flows in the DF.

**Figure 5: Multiple flows**

Besides the multiple flows, another important characteristic of printer design projects, that is difficult to represent in the DF layers, are the project phases. To reduce the risks associated with the design of very complex products, the design process is organized in phases. Each phase is geared towards the design of a physical prototype which is increasingly more complete and more alike the final product. During the first phases, these prototypes are built to evaluate the feasibility of the design and solve design issues. However, during the mid phases, these prototypes are built to define the product and solve engineering issues. During the final phases these prototypes become the actual product.

Therefore, it is also necessary to find a way to represent the different phases of the design process in the DF. Each of the phases can have several teams that work in one of the sub-systems or aspects of the prototype, and as a consequence have multiple flows. It is important to note that the phases may overlap in time. For example, sometimes there are teams working in the last design steps of one phase while other teams are working in the first design steps of a new phase. A graphical representation of the different design phases of a project can be seen in Figure 6.
Figure 6: Design phases of a project

As a conclusion, the three layers are not enough to describe the design process and it is necessary to add support for the design projects, phases and multiple flows in the DF conceptual model and prototype.

Is any change necessary in the flow layer?

Previously, the need for multiple flows was identified. However, the flows are not totally independent from each other. The models and parameters in one flow often depend on models and parameters in another flow. For example, the total productivity of the printer, determined in the productivity flow, depends on the speed at which the paper is separated, determined in the paper input flow. Therefore, the flow layer should be modified to allow the existence of these dependencies between the system designs in different flows, as shown in Figure 7.
An important consideration is that the dependencies should be propagated along when a new design step is made in one of the flows either automatically or on user’s choice depending on the context.

**Is any change necessary in the view layer?**

Over the last years, the printer manufacturing company spent some effort to determine a conceptual model to represent their system. A PhD project [9] was carried to build such conceptual model with the help of several architects. That conceptual model is called Architecture Model (AM) and reflects the major aspects of the printer system architecture. The Architecture Model pursues a similar goal as the view and model layer of the DF. Therefore, it is quite relevant to compare the key elements in the AM conceptual model to the key elements in the DF conceptual model and establish if some changes are needed in the DF conceptual model.

In the following table, the AM elements that are covered in the view layer are analyzed. Their meaning is introduced and their mapping to the DF elements, when the mapping is possible, is explained. Given that the AM does not provide an equivalent to the flow layer; the same comparison is later done only for the model layer.

<table>
<thead>
<tr>
<th>AM element</th>
<th>Meaning</th>
<th>Possible Mapping in the DF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Description of an objective or function of the system from a subjective point of view.</td>
<td>Can be represented with a Block.</td>
</tr>
<tr>
<td>Function composition</td>
<td>Function that has a sub-function.</td>
<td>Can be represented with a Block composition.</td>
</tr>
<tr>
<td>Function relation</td>
<td>Transition among functions used to describe behavior (i.e., sequences of functions).</td>
<td><strong>Not possible to establish a sequence between blocks in DF.</strong></td>
</tr>
<tr>
<td>Entity</td>
<td>A system object (software or hardware) which performs functions.</td>
<td>Can be represented with a Block.</td>
</tr>
<tr>
<td>Entity relation</td>
<td>A relation among entities, entailing transfer of energy, information, or mass.</td>
<td>Can be represented with parameter dependencies between parameters in dif-</td>
</tr>
</tbody>
</table>
Table 3: Architecture model - DF mapping

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
<th>AM Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Used to arrange information in a pictorial representation to provide an explanation.</td>
<td>Similar to the View concept of the DF. However, the elements in AM do not belong to a View.</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description of an objective or constraint related to values of parametric information.</td>
<td>Can be represented with a Parameter.</td>
</tr>
<tr>
<td>Formula</td>
<td>Mathematic representation (equation, inequality, etc.) of a relation among parameters.</td>
<td>Can be represented with a Law and a Formula dependency in DF.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Representing any variable or parameter which can be used to quantify a property. These objects can be mapped (directly or indirectly) to other objects allowing building concrete descriptions.</td>
<td>Can be represented with a parameter or block for aggregation in DF.</td>
</tr>
<tr>
<td>Entity – Parameter relation</td>
<td>A parameter that belongs to an entity.</td>
<td>Can be represented by a Parameter contained in a Block.</td>
</tr>
</tbody>
</table>

Due to the depth of the research that led to the DF conceptual model it is not surprising that a mapping between most of the elements can be easily established.

However, one important difference found in the mapping is that the DF only allows composition relationships between the system blocks. Sometimes it is necessary to have plain relations between system blocks to model the printer design. A practical example of one of those associations can be the indication of a sequence between a block indicating the paper separation function and another block indicating the printing function.

Another important difference found in the mapping is that the elements in AM have a more defined meaning in the printer manufacturing company terms (function, entity or requirement) than their DF equivalent. From a designer perspective this is important because it allows an easier understanding of the model. As a consequence it is desirable to have a way to qualify blocks in the DF with a more restrained meaning.

Is any change necessary in the model layer?

As it was previously mentioned, one of the Octo+ goals is to streamline the development process by using a federation of interconnected models. The DF model layer helps in this task by allowing the addition of models and their characterization through the input and output parameters. Nevertheless, the DF connection to the external models is weak and it relies on the designer to retype the essential parameters in the DF. The retyping of the parameters is a source of errors and a potential cause for information inconsistency. Hence, such a weak integration poses a risk for the adoption of the tool.

A stronger integration of the models was also a point analyzed in the AM. In this tool, there is a layer called external communications that enables the connection to external models. This layer allows external model elements, to be mapped to parameters, and to be imported and exported from the external models. The operations on the external model elements, represented as domain entities in the conceptual model, are done through the synthesis methods.
This kind of strong integration is very desirable in order for the DF to be successful for the design of high-tech printers. It should be possible to import and export information from the design framework to external models and to run these external models by specifying the input and output parameters in the DF.

Is the prototype usable for a designer?
The DF prototype has a number of usability problems that makes it difficult to work with it in a production environment:

- It is difficult to locate information. In large systems like a printer the number of parameters, models and experiments in the View diagram is very high. An experiment ran in the printer manufacturing company shows that, with only a small part of the total design information entered in the DF, it becomes difficult to locate elements in the prototype. As a consequence, it is necessary to provide a mechanism to easy locate information in the prototypes.
- It is not possible to visualize the relationship between different blocks, parameters, experiments and models. Some stakeholders complained that this information is mostly hidden in the properties window. A mechanism to visualize the elements and their relations is also desirable in the DF.
- It is not possible in the graphical editors to reorder elements in the View diagram. Once a parameter or a block is created it cannot be moved to another block. Sometimes when entering the information in the diagram it is necessary to do so. For example, in the case of a physical view, where each block is a part of the printer, the order of the parts is important and changes regularly during the evolution of the design. Without the reordering of elements the designer has to delete and recreate all the elements. This takes a lot of time and is prone to errors. As a consequence, it is necessary to provide a reordering of elements mechanism in the prototype.

3.3 Project Goal

It is clear that the DF conceptual model and prototype need to be adapted to be used in the design of production printers. The main goal of this project is to provide the necessary adaptations to the DF. These adaptations should help the designer to connect all the models and design information produced during the design process. It is also a goal to further validate the DF research with an industrial case.
4 System Requirements

Abstract – In this chapter, the requirements for the system are described. The requirements are based on the problem analysis detailed in the previous chapter. First the scope of the project is narrowed to some of the problems previously described. Next, the requirements are introduced and later further detailed from the user point-of-view using use cases.

4.1 Scope
During the problem analysis, several needs for the DF were discovered. It is not possible to solve all those needs in the duration of this project. Therefore, the scope of this project is limited to provide a full design and implementation for the project, phases, multiple flows and the model integration problems, and a first prototype for the relationship exploration and search problems.

For this reason, the requirement section specifies just the problems in the limited scope and the use cases and design sections present information only for the project, phases, multiple flows and the model integration problems.

4.2 Functional Requirements
In this section the problems identified are translated into functional requirements. These functional requirements are later explained in more detail and with pictures in use cases.

4.2.1 Project, phases and multiple flows
The DF should provide the designers with elements to describe the design projects. The new elements needed are:

- A new element to describe a project. This element should contain a name and the phases that are part of that project. This element should be visualized in a new editor called Project editor. It should also be possible to open one of the phases from the Project editor.
- A new element to describe a phase. This element should contain the name, the flows for the different sub-systems or aspects designed during that phase, and a link to a previous phase if it exists. This element should be represented visually in the Project editor and edited in a new editor called Phase editor. It should also be possible to copy information from a previous phase.

The DF should also provide the designers with elements to describe the multiple flows and their relations. The new elements needed are:

- A new element in the Phase editor to represent the multiple flows. This new element should represent the same flow that can be edited with the Flow editor. A natural way to visually represent the multiple flows is with swimlanes.
- A new element to represent the dependencies between the design steps in different flows. A natural way to represent this dependency is with a link between design decisions in different flows. This link is further called a flow dependency.
The flow dependencies between two different flows X and Y should have the following behavior to successfully synchronize the information contained in the partial designs specified in these flows:

- If a flow dependency exists from a design step X(n) in flow X to a design step Y(m) in flow Y then all the parameters and blocks in the system design contained in X(n) can depend on blocks and parameters on the system design contained in Y(m).
- It is possible to delete a flow dependency between a design step in X and a decision in Y if and only there do not exist design dependencies between the system designs of these design steps.
- Only one flow dependency can exist from a design step in X to a design step in Y.
- If a flow dependency exists from a design step X(n) to a design step Y(m) and a new design step X(n+1) is created from the design step X(n), then all the design dependencies existing from X(n) to Y(m) should be copied to exist from X(n+1) to Y(m) and a new flow dependency should be automatically created from X(n+1) to Y(m).
- If a flow dependency exists from a design step X(n) to a design step Y(m) and a new design step Y(m+1) is created from the design step Y(m), then the user is responsible to migrate the flow dependency from Y(m) to Y(m+1). When the user migrates this flow dependency, then the design dependencies existing from X(n) to Y(m) should be migrated to exist from X(n) to Y(m+1).

4.2.2 Model integration

The DF should allow external models to be integrated. The DF should be able to understand the formalisms of the external models and perform operations on them through an interface called model provider.

The DF should provide the designers with a new kind of element to represent models that can be integrated with external models. This new element can be associated with a provider that understands the formalism of an external model and is able to perform one or more of the following operations:

- Import blocks and parameters from the external model into the DF.
- Export blocks and parameters from the DF to the external model.
- Run the external model by passing the input parameters defined in the DF and retrieving the output parameters after the execution of the model with the updated input parameters.
- The model integration solution should be independent from the interface, and it should be possible to execute the model integration operations from an editor or a command line interface.

4.2.3 Search

The DF should allow the user to search for information. It should be possible to search for a string or a regular expression. It also should be possible to filter the search based on the type of element being searched (design decision, view, parameter, block, model or experiment).

The search should be performed in the meta-model and it should provide a list of elements that match the string or regular expression being searched. It also should be possible to select one of the elements in the search result and navigate to the diagram that contains the element.
4.2.4 Relationship exploration
The DF should allow the user to visually explore the different relations between blocks and parameters. This exploration should allow the user to:

- Explore the relationships from a block to other related blocks
- Explore the relationships from a parameter to other blocks and parameters.

In order to do that, the DF should provide a new type of editor called relationship exploration editor.

4.3 Non-Functional Requirements
The following general non-functional requirements were identified:

- It should be possible to use the DF without the new company specific functionality.
- It should be possible for several simultaneous users to use the DF through SVN.
- Errors should be displayed in a user understandable way in the user interface.

In addition some non-functional requirements were identified for the model integration problem.

4.3.1 Model integration Non-Functional requirements

- The model integration should support new types of models in the future.
- It should be possible for a printer designer to add a new model provider to his/her DF installation from a model provider repository as it is shown in Figure 8.

Figure 8: Model Provider Repository

- The command line interface solution should execute between a few seconds after it is invoked as it might be used in a build process.

4.4 Constraints
The following general constraints were identified:

1. The new functionality should be based on the DF existing prototype, thus following the implementation structure and the design philosophy and using Eclipse Modeling Framework and Graphical Modeling Framework.
4.5 Use cases: Project, phases and multiple flows

4.5.1 Create a project diagram

Brief Description
The user creates a new project diagram in the DF and opens it with the Project editor.

Flow of events
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)

1. U: Selects “Create a new project diagram” in the associated wizard.
2. DF: Prompts for the following data: project diagram name, location and model name and location.
3. U: Inserts the requested data.
4. DF: Creates the new project diagram file and the new model file. Both the diagram and the model are initially empty.
5. DF: Opens the project diagram with the Project editor.

Figure 9: New project diagram created and opened

4.5.2 Add a new phase

Brief Description
The user creates a new project phase in a project diagram using the Project editor.

Flow of events
Level: User-goal level
Primary Actors: User (U); The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-Condition: A project diagram is open.

1. U: Selects “Phase” in the palette and drops it in the diagram displayed in the Project editor.
2. DF: Prompts for the following data: phase name.
3. U: Adds the phase name and specifies a previous phase if necessary.
4. DF: Creates a new Phase element in the model and updates the diagram and model files.
4.5.3 Add a flow

**Brief Description**
The user creates a new flow in the Phase editor.

**Flow of events**
Level: User-goal level  
Primary Actors: User (U). The user is an architect or a designer.  
Scope: Design Framework (DF)  
Pre-condition: A phase element was added to the project and the Project editor is open.

1. U: Opens a phase diagram by selecting a phase visual element in the Project editor.  
2. DF: Shows the phase diagram in the Phase editor.  
3. U: Selects “Flow” from the palette and adds it in the Phase editor.  
4. DF: Prompts for the flow name.  
5. U: Inserts the name  
6. DF: Creates the flow element, an initial design decision element with an initial system design element and updates the diagram and model files.
4.5.4 Open a flow from the Phase editor

**Brief Description**
The user opens a Flow editor from the Phase editor.

**Flow of events**
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)

**Basic Flow**

1. U: Selects a flow visual element in the phase diagram.
2. DF: Displays a menu with the action “Open Flow Diagram”.
3. U: Selects that action.
4. DF: Creates a model repository, and a new flow diagram and model file. The design decision and system design elements created before are added to this model file and shown in the diagram.
5. DF: Opens the new flow diagram with the Flow editor.
6. U: Uses this diagram to work with the flow as in the core version of the Design Framework.

**Alternative Flow**
At point 4. If the flow diagram was already created then it just opens the created diagram.

4.5.5 Add a flow dependency

**Brief Description**
The design information in flow depends on the design information in another flow.

**Flow of events**
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Post-Condition: From a system design in the linked decision is now possible to create parameter dependencies and to use models from the system design in the linked design decision.

**Basic Flow**

1. U: Selects the “Flow Dependency” creation tool in the palette of the Phase editor. Then it selects a source design decision and a target design decision.
2. DF: Creates a flow dependency between the two design decisions and show an arrow.

**Alternative Flow**
At point 2. When the flow dependency is created the rules defined in the requirements for Project, phases and multiple flows (in section 4.2.1) should be followed.
4.6 Use Cases: Model integration

4.6.1 Attach an external model

Brief Description
The user attaches an external model to the DF.

Flow of events
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-condition: A model element was added to a block and the model file that contains the model was specified.

Basic Flow

1. U: Selects a model visual element in the view diagram using the View editor.
2. DF: Fills and displays a menu with the possible actions that can be executed.
   The action list depends on the model providers registered for the selected model element and the extension of the model file.
3. U: Selects the action “Attach the model with provider x” from the menu.
4. DF: Executes the model provider that “understands” the formalism of the model and parses and displays a dialog with the model internal structure, the external model information and the mappings created along the way.
5. U: Creates a match for the external parameters in the model with the internal parameters in DF previously defined in the View editor and presses “ok”.
6. DF: Creates the model elements to hold a mapping and store the provider that was used for attaching the model. It also marks the model as attached.

Alternative Flow

At point 5. If the model provider supports the import operation, then it is also possible to copy blocks and parameters from the external model to the DF. In that case the match between the external parameters and the internal parameters is automatically created when the blocks and parameters are copied to the DF.

At point 5. If the model provider supports the export operation, then it is also possible to copy blocks and parameters from the DF to the external model. In that case the match between the internal parameters and the external parameters
is automatically created when the blocks and parameters are copied to the external model.

Figure 13: Attach a model

4.6.2 Run an external model

**Brief Description**
The user executes an analysis run of an external model and stores the conditions and results in an experiment.

**Flow of events**
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-condition: A model was attached to the DF and the external parameters of the model were mapped to internal parameters.

**Basic Flow**

1. U: Selects a model visual element in the view diagram using the View editor.
2. DF: Fills and displays a menu with the possible actions that can be executed. The action list depends on the model providers registered for the selected model element.
3. U: Selects the action “Run the model with provider x” from the menu.
4. DF: Takes the value of the mapped input parameters and passes them to the model provider. The model provider communicates with the external model by providing the parameters retrieving the result. When the result parameters become available, the model provider copies their value to the mapped output parameters so the value in the DF of coupled parameters are always those of the last experiment that was done. For each run action it also creates a new experiment where the input parameters and the output parameters are stored.
4.6.3 Import information from an external model

**Brief Description**
The user imports information from an external model.

**Flow of events**
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-condition: A model was attached to the DF and the external parameters of the model were mapped to internal parameters.

**Basic Flow**
1. U: Selects a model visual element in the view diagram using the View editor.
2. DF: Fills and displays a menu with the possible actions that can be executed. The action list depends on the model providers registered for the selected model element.
3. U: Selects the action “Import the model with provider x” from the menu.
4. DF: Copy the value of the external parameters, using the external model through the model provider, into the DF mapped parameters.

4.6.4 Export information to an external model

**Brief Description**
The user exports information to an external model.

**Flow of events**
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-condition: A model was attached to the DF and the external parameters of the model were mapped to internal parameters.

**Basic Flow**
1. U: Selects a model visual element in the view diagram using the View editor.
2. DF: Fills and displays a menu with the possible actions that can be executed. The action list depends on the model providers registered for the file extension of the selected model element.
3. U: Selects the action “Export the model with provider x” from the menu.
4. DF: Copy the value of the DF parameters into the mapped external parameters, resulting in the external model being changed.

4.6.5 Register a model provider

**Brief Description**
The Design Framework developer creates a new model provider

**Flow of events**
Level: User-goal level
Primary Actors: DF developer (D), User (U).
Scope: Design Framework (DF)

**Basic Flow**

1. D: Creates a new model provider that implements the model provider interface distributed with the DF.
2. D: Deploys the model provider to the model provider repository.
3. U: Install the model provider from the model provider repository in his DF installation.
4. DF: Register the model provider in the DF so that it can be used.

4.7 Use Cases: Search

4.7.1 Search for an element

**Brief Description**
The user searches for an element in DF.

**Flow of events**
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)

1. U: Selects the search option.
2. DF: Displays an empty search dialog.
3. U: Enters the search information: string to search and optionally the type of element requested.
4. DF: Displays the list of resulting elements that match the string. If the elements are present in multiple diagrams, then they are collapsed under one element in the results that can be expanded to show the diagrams in which it is present.

![Search option](image)

Figure 15: Search option
4.7.2 Select an element resulting from the search

Brief Description
The user selects a resulting element from the search result list.

Flow of events
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-Condition: A search was done and the list of resulting elements has one or more elements.

1. U: Selects one of the elements from the list.
2. DF: Displays a menu with the action “Open the diagram and highlight the element”.
3. U: Selects the action.
4. DF: Open all the diagrams where the selected element is present.

Alternative Flow
At point 2. When the element is present in multiple diagrams and if the element is collapsed, then the action opens all the diagrams.
At point 2. When the element is present in multiple diagrams and if the element is expanded, and if only one of the diagrams is chosen then the action opens only that diagram.

4.8 Use Cases: Relationship exploration

4.8.1 Open the relationship exploration

Brief Description
The user performs an exploration of the relationships between elements in DF.

Flow of events
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-Condition: A View diagram is open and has some element inside.

1. U: Selects a block or parameter in a view diagram using the View editor.
2. DF: Displays a menu with the action “Open Relationship Exploration Diagram”.
3. U: Selects the action.
4. DF: Creates the diagram and adds a shortcut in the diagram to the selected element where the exploration should start from.

4.8.2 Explore

Brief Description
The user performs and exploration of the relationship.

Flow of events
Level: User-goal level
Primary Actors: User (U). The user is an architect or a designer.
Scope: Design Framework (DF)
Pre-Condition: A relationship exploration diagram is open using the relationship exploration editor.

1. U: Selects one of the visual elements present in an exploration diagram.
2. DF: Displays a menu with the actions “Explore up”, “Explore down”, “Explore backward” and “Explore forward”.
3. U: Selects one of the actions.
4. DF: Displays the elements that result from the exploration and does a basic layout action.

Alternative Flow
At point 1. The user can select to re-order all the elements automatically with the layout action.

Figure 16: Relationship exploration
5 System Architecture and Design

Abstract – In this chapter the system architecture and design are provided based on the requirements introduced in the previous chapter. Various architectural views are presented in order to explain the design of the system.

5.1 Introduction

This chapter provides a comprehensive architectural description of the solution. This description is based on the main design decisions, an architecture overview and the “4+1” architecture model [10], [11]. This model introduces four architectural views to describe the different aspects of the system that concern different stakeholders and an extra view with the use cases. In each of these views the most significant artifacts and architectural decisions are detailed. These views are:

- The Logical view which describes the system from a conceptual point-of-view.
- The Development view which describes the system from an implementation point-of-view.
- The Deployment view which describes the system installation and execution in a network of computers.
- The Process view which describes the system processes and their communication.

The following sections are described with the help of UML diagrams and architectural and design patterns. More information about these patterns can be found in [12] and [13]. Additionally, in this and the following sections the existing DF is referred to as DF Core and the solution introduced in this project is referred to as DF Octo+.
5.2 Main design decisions
Several non-functional requirements were introduced. Some of these requirements led to important architectural decisions. Following is a description of these decisions. These decisions are further detailed in the four architectural views.

5.2.1 Design Decision: Expand the DF
An important design decision was to add the new company specific functionality as a set of extensions to the DF Core components. The rationale of this decision is explained in the following table.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>It should be possible to use the DF Core without the new company specific functionality.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternatives</td>
<td>i. Modify the DF Core with the new functionality and decide which functionality to use based on configuration.</td>
</tr>
<tr>
<td></td>
<td>ii. Copy the DF Core and create a new independent product that has both the old and the new functionality.</td>
</tr>
<tr>
<td></td>
<td>iii. Expand the DF Core, without modifying it, and add the new functionality on top of it.</td>
</tr>
<tr>
<td>Selected alternative</td>
<td>Expand the DF Core by introducing a set of extensions that extend the DF Core components.</td>
</tr>
<tr>
<td>Rationale</td>
<td>i. In the first alternative it would be necessary to modify the DF Core for each company that uses it. In that case, the design would quickly become incoherent and difficult to maintain.</td>
</tr>
<tr>
<td></td>
<td>ii. In the second alternative there would be a lot of duplicated code that would cause maintainability and evolution problems.</td>
</tr>
<tr>
<td></td>
<td>iii. The third alternative has many advantages that are detailed below.</td>
</tr>
</tbody>
</table>

**Maintainability:** The code is not duplicated and the design of company specific features is separated from the DF Core components design. Whenever the DF Core components are changed without breaking the interface, the changes are propagated to all the extensions using them. In addition, in the best case the maintenance staff does not need to know all the specific products and their parts.

**Evolution:** The company specific extensions can be made part of the DF Core or become company independent extensions if they prove to be reusable in different contexts.

**Quality:** The DF Core quality is assured only once and reused in all the products.
5.2.2 Design Decision: Extendable model integration

Another important decision was to provide an extensibility mechanism for the model integration. The rationale of this decision is explained in the following table.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>The model integration should support new types of external models.</th>
</tr>
</thead>
</table>
| Alternatives| i. Provide the model integration through an intermediate format to which the external models can be converted to.  
                   ii. Provide the model integration through an interface that can be implemented so that the DF Octo+ extension can “understand” the external models formalism, and a plug-in architecture to add the implementation to these interfaces dynamically to the DF. |
| Selected alternative | Provide the model integration through an interface, called model provider, and a plug-in architecture. |
| Rationale | The second alternative includes the first one. An implementation can be the integration through an intermediate format. It is also more flexible as it allows a deeper integration and the possibility to run analysis on the models in real time. |
5.3 Architecture overview

The DF Core is based on three components: a meta-model, the Flow editor and the View editor. The meta-model represents the conceptual model introduced in Chapter 3 and the editors are used to create and edit the elements defined in the meta-model. A technology called Eclipse Modeling Framework (EMF) was used to create the meta-model and another technology called Graphical Modeling Framework (GMF) was used to create the editors.

As it was previously described, the solution is based on a set of extensions on top of the DF Core components. These extensions are: The DF Octo+ meta-model, the Project editor, the Phase editor, the model integration, the Flow editor contribution and the View editor contribution. The DF Octo+ meta-model extends the DF Core meta-model, the Project and Phase editors are used to edit some of the elements in the extended meta-model and the Flow and View extensions extend the Flow and View editors to use the elements in the extended meta-model. EMF and GMF were also used to create the DF Octo+ meta-model and editors.

The main components of the DF Core and the DF Octo+ and their relation can be seen in Figure 19. The dependencies to GMF and EMF have been omitted to make the diagram more readable. These components are detailed from a conceptual, development, deployment and process perspective in the following sections.

Figure 19: Architecture overview
5.4 Logical View
The logical view is concerned with the functionality that the system provides to end-users, supports functional requirements and shows how the system is decomposed into a set of abstractions.

5.4.1 DF Core meta-model
The DF is a visual modeling tool used to describe system designs and the rationale behind them. The models created with it, referred to as DF models, are based on a meta-model that formally defines the modeling elements and their relations. These elements, based on the conceptual model introduced in Chapter 3, are called semantic elements and the most important one are described in Table 4 and Table 5 and in the diagrams shown in Figure 20 and Figure 21.

![Diagram](image-url)
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignFlow</td>
<td>The DesignFlow element represents the design process composed of several design decisions.</td>
</tr>
<tr>
<td>DesignDecision</td>
<td>The DesignDecision element represents the design decisions composed of questions and answers and a system design.</td>
</tr>
<tr>
<td>SystemDesign</td>
<td>The SystemDesign element represents the system under design at a certain point in time and is composed of several design views.</td>
</tr>
<tr>
<td>QuestionandAnswer</td>
<td>The QuestionandAnswer element represents the questions answered in each design decision through models and experiments.</td>
</tr>
<tr>
<td>DesignView</td>
<td>The DesignView element represents the different aspects that describe a system design and is composed of system blocks.</td>
</tr>
<tr>
<td>System Block</td>
<td>The SystemBlock element represents the parts of the system under design. The SystemBlock element is characterized by experiments, models and parameters.</td>
</tr>
<tr>
<td>Model</td>
<td>The Model element represents the models made by the architects and designers to answer to the design questions and take a design decision.</td>
</tr>
<tr>
<td>Experiments</td>
<td>The Experiment element represents the analysis made of a model.</td>
</tr>
</tbody>
</table>

Table 4: DF meta-model elements part 1

![Diagram of Design Flow, Design View, and Model](image)

Figure 21: DF meta-model part 2
5.4.2 DF Octo+ meta-model

The new elements of the DF Octo+ meta-model extension are described in Tables 6 and 7 and in the diagrams shown in Figures 22 and 23. In the diagrams, the new semantic elements added to the meta-model have a colored background.

As it was previously described, an important decision was to extend the DF Core meta-model. Therefore, as it can be seen in the diagrams, only the DF Octo+ elements have dependencies to the DF Core elements. The DF Core elements remain independent from the added elements.

The first diagram describes the new meta-model elements added to represent the project, phases and multiple flows. The rationale and requirements to introduce these new elements were described in Chapter 3 and Chapter 4.
Figure 22: Project, phases and multiple flows meta-model

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignProject</td>
<td>The DesignProject element represents a printer design project in that is composed of several phases.</td>
</tr>
<tr>
<td>DesignPhase</td>
<td>The DesignPhase element represents one of the project phases in printer design process and has several flows thus representing the multiple flows nature of the phases. The phases can have also previous phases.</td>
</tr>
<tr>
<td>FlowContainer</td>
<td>The FlowContainer element represents a container for a flow. It allows introducing changes to the flow without modifying the element. The FlowContainer can be related to another FlowContainer in another phase. This relation represents that the flow is based on a flow in a previous phase, probably done by the same person or team.</td>
</tr>
<tr>
<td>LinkableDesignDecision</td>
<td>The LinkableDesignDecision element represents a design decision that can be connected to other design decisions in other flows. When a LinkableDesignDecision is connected to another LinkableDesignDecision, the two decisions can have dependencies between the parameters in their system designs.</td>
</tr>
</tbody>
</table>

Table 6: Project, phases and multiple flows meta-model elements
The second diagram describes the elements added to represent the model integration. For the model integration, the Model element was extended to be integrated with external models. The structure of these external models is represented with the ExternalAttachedModel, the ExternalAttachedBlock and the ExternalAttachedParameter elements. The integration between the external parameters and the parameters in the DF is represented with the IntegratedParameter elements.

An important design decision regarding the model integration meta-model was to represent the external model structure on it. An alternative was to build a transient structure every time that an operation on the external model is performed. However, by representing the external model structure in the meta-model it is possible to store the structure in the model instances, to edit the external structure from the DF editors and to easily maintain the DF in a consistent state when the external model is changed or is not accessible.

Figure 23: Model integration meta-model
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntegratableModel</td>
<td>The IntegratableModel element represents a model that can be integrated with external models so that it can be run from inside the DF or that parameters can be imported or exported. When the model is integrated then the IntegratableModel has a list of the integrated parameters.</td>
</tr>
<tr>
<td>ExternalAttachedModel</td>
<td>The ExternalAttachedModel element represents an external model.</td>
</tr>
<tr>
<td>ExternalAttachedBlock</td>
<td>The ExternalAttachedBlock element represents a block in the external model.</td>
</tr>
<tr>
<td>ExternalAttachedParameter</td>
<td>The ExternalAttachedParameter element represents a parameter in the external model.</td>
</tr>
<tr>
<td>IntegratedParameter</td>
<td>The IntegratedParameter element represents a mapping between a parameter internal to the DF and an external parameter. When this mapping exists, then it is possible to import and export the values between the DF and the external model.</td>
</tr>
</tbody>
</table>

Table 7: Model integration meta-model elements

5.4.3 Editors
The meta-model define the DF semantic elements and their relationships. These elements need to be visually created and edited. For this reason, it is necessary to have some editors that have the following functionalities:

- Create new diagrams, save and reopen diagrams.
- Create visual representations of the semantic elements.
- Visually edit these semantic elements.

The editors are an essential part of the DF Core and the DF Octo+ extensions. The DF editors are complex components created using a technology called GMF. In this section the editors’ structure and behavior is introduced from a conceptual point of view and the main abstractions are described. In the development view more information is added on the design of the editors.

Conceptually, an editor is composed of a Canvas and a Palette. The Canvas is where the diagram is edited and the palette is used to add visual elements to the canvas. The diagram and the visual elements are the parts of the editor that are used to visually edit the SemanticElements, and as a consequence they can be called EditParts. The DiagramEditPart and the corresponding SemanticElements are created at the same time as the diagram. The rest of the EditParts and SemanticElements are created through the Palette with the CreationTools.

The class diagram in Figure 24 shows the conceptual structure of an editor.
5.4.4 The DF Core editors

In the DF core there are two editors, the Flow editor and the View editor. The Flow editor is used to edit the following semantic elements:

- DesignFlow
- DesignDecision
- QuestionAndAnswer
- SystemDesign
- DesignView

The View editor is used to edit the following semantic elements:

- DesignView
- SystemBlock
- Parameter
- Dependencies
- Model
- Experiment

The Flow and View editors are connected with each other through the DesignView semantic element that can be edited in both editors.

5.4.5 The DF Octo+ editors

As it was previously described in Chapter 4, one of the main functional requirements is to add new editors to visually edit the new semantic elements. Two new editors are added in the DF Octo+, the Project editor and the Phase editor. The Project editor is used to edit the following semantic elements:

- DesignProject
- DesignPhase

The Phase editor is used to edit the following semantic elements:

- DesignPhase
- DesignFlowContainer
- DesignFlow
The Project and Phase editors are connected to each other through the DesignPhase semantic element that can be edited in both editors, and the Phase and Flow editors are connected to each other through the DesignFlow semantic element.

5.4.6 Model integration

As it was also mentioned, another important functional requirement is the model integration. For the model integration it is necessary to extend the View editor to execute the model integration operations on the ModelEditPart. The model integration conceptual structure is shown in Figure 25.

The ModelEditPart, the EditPart used to visually edit the Model semantic element, is responsible to receive the user request for the four model integration operations (attach, import, export and run) and to coordinate the execution of the operations with the other abstractions. The MatchingDialog is responsible for the creation of the mapping to the external model and for the copy of information between the DF and the external model. The ModelProvider is responsible for the connection to the external model and for the creation of the meta-model representation of the external model. The model integration dynamic aspect can be seen in the sequence diagram in Figure 26.

The first step towards integrating an external model is to attach it to the DF. To attach the model, the user executes the attach operation of the ModelEditPart, in the attach operation, the external model is imported and the MatchingDialog is created.

Importing a model means to obtain the external model in the DF representation of the external model. This representation is called ExternalAttachedModel. It is necessary to have an internal representation of the external model because the DF needs to be connected to many different external models. The ModelProvider is responsible for converting between the specific formalisms of the external models to the ExternalAttachedModel.
Once the DF knows the external model representation, the user can copy Blocks and Parameters (part of a system design) from the DF to the external model representation and vice versa. When a parameter is copied it is also integrated. An IntegratedParameter is a mapping between a Parameter and an ExternalAttachedParameter.

After an external model is attached, the model containing it can be run, exported or imported as it can be seen in Figure 26.

- When the model is exported, the value of the integrated Parameters is copied to their corresponding ExternalAttachedParameter and the ExternalAttachedModel is passed to the ModelProvider to convert it to the specific formalism.
- When the model is imported, the ExternalAttachedModel is obtained from the ModelProvider and the value of the ExternalAttachedParameter is copied to the corresponding integrated Parameter.
- When the model is run, a similar behavior to an import/export happens with the difference that an experiment is externally executed in the model. This experiment takes the value of some input Parameters and obtains as a result the value of some output Parameters.

![Figure 26: Model integration dynamic structure](image)

5.5 Development view

In the development view, the architecture is described from an implementation point-of-view. The conceptual elements introduced in the logical view, are further elaborated based on the technological constraints and the non functional requirements.
5.5.1 EMF meta-model generated classes

As it was previously described the solution needs to be based on EMF. The EMF is a model-driven technology used to generate java classes based on meta-model definitions. The meta-model definitions are done using a custom UML version called ECore.

The EMF generated classes have a strong design that imposes a big constraint in the design of the rest of the solution. Therefore, it is important to describe the EMF and EMF generated classes design. However, because of the scope of this work, only the aspects of the EMF and EMF generated classes design that affect the solution are described. A complete description of EMF benefits and design can be found in [14].

EMF generated classes design

EMF generates four categories of classes from the meta-models that are organized in four different packages. This package decomposition can be seen in Figure 27. The names of the packages can be changed in the generators.

![Figure 27: EMF generated packages decomposition](image)

The model package contains the meta-model elements interfaces and the factory interface to create these meta-model elements. An important decision in EMF generated classes is to create interfaces and implementations for all the meta-model elements. The model.impl package contains the classes that implement these meta-model elements interfaces and the implementation of the factory defined in the model package.

Every generated meta-model element in EMF extends the EMF type EObject. By extending this type the meta-model elements also extend the Notifier type. A Notifier is equivalent to a Subject in the observer pattern: it can notify Observers of changes in the state. The Observers in EMF are called Adapters, because besides listening to changes in the state of the observed objects they are used to extend the behavior of these observed objects by adapting the notifications to external interfaces.

The provider package contains generated Adapters for the meta-model elements. These Adapters are called Item Providers and every meta-model element has an associated Item Provider. The Item Providers are Observers of the meta-model classes and Adapters to certain interfaces. As Adapters, they adapt the meta-model classes to interfaces needed by clients, typically views and editors. As Observers, they are notified of all the changes in the meta-model classes that they can translate into the clients’ interfaces.

The editor package contains classes that implement a UI editor to visually edit the meta-model elements and create the meta-model files. In order to do so the editor
classes use the Item Providers classes. The editors are implemented to be used in the Eclipse Platform. As a consequence, the editor package contains two generated classes that implement the interfaces IEditorPart and INewWizard. These interfaces belong to the Eclipse Platform, and by implementing these interfaces the EMF UI editor can be used in Eclipse.

5.5.2 DF meta-model generated classes

The generated classes for the DF meta-model described in Section 5.4.1 are organized under the package decomposition described in Figure 27 but with the package names prefixed with \textit{nl.esi.design}. This package decomposition can be seen in Figure 28.

![Figure 28: DF meta-model package decomposition](image)

5.5.3 DF Octo+ meta-model generated classes

The generated classes for the DF Octo+ meta-model described in Section 5.4.2 are also organized under the package decomposition described in Figure 27 but with the package names prefixed with \textit{nl.esi.design.octoplus}. This package decomposition can be seen in Figure 29.

![Figure 29: DF Octo+ meta-model package decomposition](image)

As the elements in the DF Octo+ meta-model use and extend the elements in the core meta-model, the packages that contain the generated classes for the DF Octo+ meta-model elements have dependencies to the packages that contain the generated classes for the core meta-model elements. These dependencies are shown in Figure 30.
5.5.4 GMF editors

As it was previously mentioned in Chapter 4, one of the constraints is to use GMF. GMF is a model-driven technology used to generate java classes that implement graphical editors based on the ECore meta-model definitions and graphical definitions of the meta-model elements. The GMF editors follow the conceptual editor introduced in the Section 5.4.3.

The GMF generated classes have a strong design that imposes a big constraint in the design of the rest of the solution. Therefore, it is important to describe the GMF and GMF generated classes design. However, because of the scope of this work, only the aspects of the GMF and GMF generated classes design that affect the solution are described. GMF is a very complex technology that integrates three different technologies: GEF, Eclipse Platform and EMF and adds behavior on top of them. A good description, though not very complete, of GMF benefits and design can be found in [15].

GMF generated classes

GMF generates nine categories of classes that are organized in nine different packages. From the point of view of this project only five of these packages are relevant. These five packages and their relations can be seen in Figure 31.
The `edit.parts` package contains a generated class `EditPart` and `Figure` for each meta-model element that is visually represented in the editor. GMF is based on a Model-View-Controller pattern where the EditPart classes are the Controllers, the Figure classes are the Views and the meta-model elements are the Models.

As Views, the Figure classes are responsible for displaying elements in a canvas. As Controllers, the EditPart classes are responsible for answering to requests from the UI and update the correspondent meta-model elements. In addition, they are also responsible for answering to changes in the meta-model elements and update the correspondent Figures.

The `edit.commands` package contains Command classes. GMF implements a command pattern to perform the undo/redo capability in the editors and uses the Command classes. In fact, the EditPart classes do not update the Figure classes and meta-model elements directly; instead they create the Command classes. In addition, as the same commands need to be created for many EditPart classes in some conditions, the EditPart classes delegate the creation of the Commands to the EditPolicy classes. Some of these EditPolicy classes are in the `edit.policies` package and some are part of the GMF libraries.

For example, the layout of the figures is done with the LayoutEditPolicy. All the EditParts have a default LayoutEditPolicy installed. The LayoutEditPolicy understands several requests like Move Children or Delete Children and produces Commands to affect the layout of the element according to the received request.

The `part` package contains classes that implement the UI editor. As in the case of the EMF generated editor classes, these classes implement the IEditor and INewWizard interfaces.

The `providers` package contains classes that implement some of the GMF service providers. The service providers are interfaces that the editors use to realize some of the behavior. By using these interfaces the editors can be extended without modifying the existing classes. From the point of view of this project the most relevant service providers are the EditPartProvider and the EditPolicyProvider. The editor uses the EditPartProvider interface to create the EditPart classes from the meta-model.

**Figure 31: GMF generated packages decomposition**
elements and the EditPolicyProvider to create and add the EditPolicy classes to the EditPart classes. This mechanism is explained in more detail in Section 5.5.7.

The model package, with the interfaces to the meta-model elements, is used by the providers package to initialize the EditParts, by the part package in the class that implements the INewWizard interface to create the model files and in the command packages in the Command packages to create the instances of the meta-model elements in the editor.

5.5.5 The DF editors generated classes
The generated classes for the Flow and View editors are organized under the package decomposition described in Figure 31 but with the package names prefixed with nl.esi.design.diagram.flow and nl.esi.design.diagram.view.

5.5.6 The DF Octo+ extension editors generated classes
As it was described in Chapter 4 two additional editors need to be added. These are the Project editor and the Phase editor. These editors are used to visually edit the elements defined in Table 6 and Table 7. The generated classes for the Project and Phase editors are also organized under the package decomposition described in Figure 31 but with the package names prefixed with nl.esi.design.octoplus.diagram.project and nl.esi.design.octoplus.diagram.phase.

5.5.7 The Flow and View editors extensibility
For the realization of the Project, phases and multiple flows, and the Model integration requirements two new elements were added to the meta-model to extend the behavior of the existing elements. These new elements are the LinkableDesignDecision that extends the DesignDecision and the IntegratableModel that extends the Model. The DesignDecision element is created in the Flow editor and the Model element is created in the View editor.

The Flow and View editor are part of the DF Core. However, as it was described, one important design decision was to extend the existing DF Core components. In order to do so an extension mechanism was introduced for the creation of new elements in these editors without modifying the DF Core editors existing classes.

This mechanism is explained in the following diagrams. The mechanism is illustrated for the creation of a DesignDecision element in the Flow editor; however, the same mechanism is used for all the elements.

The class diagram in Figure 32 introduces the classes that are involved in the creation of a new DesignDecision element in the Flow editor and the relation between them and with the GMF library classes. As it is shown in Figure 32, the classes belong to the packages previously introduced.
The sequence diagram in Figure 33 shows how some of these classes interact to add EditPolicies to an EditPart. As it can be seen in the diagram, the EditPart install the EditPolicies by invoking the createDefaultEditPolicies that is overridden in the sub-classes and by invoking the installEditPolicies in the EditPolicyProvider.

Therefore, by introducing EditPoliciesProviders it is possible to hook in the DF Core editors and remove existing EditPolicies and add new ones. In the diagram, this behavior can be seen in the optional fragment that is executed only if the editor is extended and a provider exists. In that case, the EditPolicyProvider removes DesignFlowSemanticEditPolicy and install the CustomDesignFlowItemSemanticEditPolicy.
The sequence diagram split in Figure 34 and Figure 35 illustrates the creation of a new DesignDecision in the Flow Editor via the EditPolicy classes.

The first part of the diagram in Figure 34 shows how the EditPart classes delegate the creation of the Command classes to the EditPolicy classes. In particular, in the alternative fragment it can be seen how the DesignFlowEditPart delegates the creation of the Command to the original EditPolicy or to the extended one depending on the existence of the corresponding EditPolicy.

**Figure 33: Add Edit Policies**
The second part of the diagram in Figure 35, shows how the command is executed to instantiate the DesignDecision or the LinkableDesignDecision element depending on the type of the command.

5.5.8 Model integration extensibility
Another important non-functional requirement is that the model integration should be extensible to be used with different kind of models. To achieve this extensibility it was decided to use a plug-in architecture.

A plug-in architecture allows clients to use implementations of interfaces defined at run-time rather than at compilation time. In order to obtain the implementations of
the interfaces the client uses a kind of Factory that can dynamically load and compile
the implementations. Generally these kind of Factories are called PluginManagers. In
the case of the model integration the client is called ModelIntegrationRegistry and
the interface is called ModelProvider. This structure can be seen in the class diagram
in Figure 36.

Figure 36: Plug-in architecture structure

The most important characteristic of a plug-in architecture is that the new implemen-
tations can be added after a product is released. As it can be seen in the sequence
diagram in Figure 37, this is possible because the implementations are dynamically
discovered, compiled and loaded from the underlying platform at run-time.

Figure 37: Plug-in architecture sequence

As a consequence new ModelProviders can be externally created just by implement-
ing an interface.

As Eclipse provides a very powerful plug-in architecture, it was also decided to reuse
that architecture instead of implementing a custom one.

5.5.9 Model integration user interface independence

In addition to the extensibility, another important non-functional requirement is that
the model integration should be independent from the user interface. This is because
it needs to be executed from the design framework editors as well as from the com-
mand line. To accomplish this, an important abstraction called Integrator was intro-
duced. The Integrator class, which is totally independent from the user interface,
works as a Façade and orchestrates the model integration.
The Integrator class performs the operations on the external model by using the ModelProvider. The ModelProvider is obtained by the EditorActionHandler, or the CommandLineApplication through the ModelProviderRegistry using the plug-in architecture previously described.

Besides the ModelProvider the interface is also composed of the three meta-model classes: ExternalAttachedModel, ExternalAttachedBlock and ExternalAttachedParameter. The implementations of the ModelProvider interface create these classes and they are used to represent the constructs of the external models in the DF. The Integrator is also responsible for converting these classes to the DF classes SystemBlock and Parameter, and for synchronizing the external model with the stored version of the external model.

The class diagram describing these classes can be seen in Figure 38.

5.6 Deployment View

The deployment view describes the topology of software components on the physical layer as well as the communication between these components.

5.6.1 Eclipse plug-ins

The DF is deployed as a set of eclipse plug-ins that can be installed in the Eclipse platform. Eclipse is not a single monolithic program, but rather a small kernel containing a plug-in loader surrounded by hundreds (and potentially thousands) of plug-ins. Each plug-in contributes to the whole in a structured manner, might rely on services provided by another plug-in, and each might in turn provide services on which yet other plug-ins may rely.

In Eclipse a plug-in is a jar component with the compiled classes and a couple of plug-in configuration files called Manifest and Plugin.xml. The configuration files define the dependencies and the services that the plug-ins provide and consume. The
plug-ins are additionally grouped in features and the features can then be grouped in products or installed, updated or removed in Eclipse through an update site in a web server.

In the DF, the packages defined in this chapter are mapped to different plug-ins and the plug-ins to features. The DF Core has two features **core** and **editors**. Following the DF Core structure it was decided to add also two new features in the DF Octo+:
- **Core**, containing the extended meta-model
- **Editors**, containing the new editors, the contributions to the existing editors and the model integration.

The DF Core and the DF Octo+ Eclipse plug-ins and features can be seen in the subsequent diagram.

**Figure 39: DF Eclipse plug-ins and features**

### 5.6.2 Typical configuration

In a typical configuration, a number of designers will be working in their own instance of Eclipse with all the DF features installed. The features are installed in Eclipse from a plug-in web server. In addition to the DF features, new features will be created to support the specific formalisms and integrated through the model integration, as described in the Model integration Non-Functional requirements section. These model integration features will also be installed from this plug-in server.

In their installation of the DF, the designers work with the DF models. The DF models are persisted in two kind of files. One to store the project information (with the extension .designoce) and another to store the flow information (with the extension .design). Therefore, in a typical configuration there is one .design model file per design team and a .designoce model file per project. These model files are also stored in a SVN server to enable the sharing of information. In addition to the designers, a build process uses the command line version of the DF to import, export or run the models.

The following diagram shows the typical configuration previously described.
5.7 Process View

The process view deals with the dynamic aspects of the system architecture and explains the system processes and how they communicate.

5.7.1 Main processes

As it was described in the previous section, the DF is executed in two main processes, in Eclipse and as a command line application. These processes access the local versions of the model files that are copied from a central SVN repository. In Eclipse, these model files are retrieved using the Subclipse Eclipse plug-in. The dynamic of the DF in Eclipse can be seen in Figure 41.
In the command line application, it is the responsibility of the process that invokes the model integration operations, typically a build process, to retrieve the model files from the SVN server. The dynamic of the command line version of the DF can be seen in Figure 42.

Figure 41: Eclipse process

Figure 42: Command line process
6 Verification & Validation

Abstract – In this chapter, the verification and validation approaches are described. The verification was done with a set of test cases, following the requirements and use cases. The validation was done with a set of real usage scenarios.

6.1 Verification

To verify the product a number of test cases were created and executed. The first level of test cases is based on the use cases and verifies that the basic behavior of the system is correct. The second level of test cases is based on corner cases and on the different errors encountered in the different releases. The second level of test cases verifies more complex scenarios.

6.1.1 Use cases verification

As the test cases based in the use cases follow the use case interaction, in the following table these test cases are only mentioned but the actions to execute them can be extracted from the use cases.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Reference</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a project diagram</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Add a new phase</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Add a flow</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Open a flow from the Phase editor</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Add a flow dependency</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Attach an external model</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Run an external model</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Import information from an external model</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Export information to an external model</td>
<td>See use case</td>
<td>Pass</td>
</tr>
<tr>
<td>Register a model provider</td>
<td>See use case</td>
<td>Pass</td>
</tr>
</tbody>
</table>

6.1.2 Additional test cases

Multiple flows

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Actions</th>
<th>Expected Result</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create two flows with the same name.</td>
<td>1. Open phase diagram.</td>
<td>The DF warns the user that a flow diagram already exists with the given name and that it must be renamed.</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Create flow with name flow 1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Create another .flow with name flow 1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Open flow diagram.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Open a flow without a name.</td>
<td>1. Open phase diagram.</td>
<td>The DF warns the user that it must specify a name.</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Create flow without a name.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Open flow diagram.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Delete a flow</td>
<td>1. Open flow diagram.</td>
<td>The DF warns</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>Task Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Create a flow in the core DF with several design decisions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Import it manually in the model file of the DF Octo+.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Create a diagram from that model file.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>Open the flow.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The flow is open correctly in the flow editor.  
Pass

<table>
<thead>
<tr>
<th>Step</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Open a phase diagram.</td>
</tr>
<tr>
<td>2.</td>
<td>Create a new flow.</td>
</tr>
<tr>
<td>3.</td>
<td>Resize it manually.</td>
</tr>
</tbody>
</table>

The swimlane is resized.  
Pass

<table>
<thead>
<tr>
<th>Step</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Open a phase diagram.</td>
</tr>
<tr>
<td>2.</td>
<td>Add a flow.</td>
</tr>
<tr>
<td>3.</td>
<td>Open the flow diagram.</td>
</tr>
<tr>
<td>4.</td>
<td>Add a very long Q&amp;A in the initial decision.</td>
</tr>
<tr>
<td>5.</td>
<td>Go back to the phase diagram.</td>
</tr>
<tr>
<td>6.</td>
<td>Click in the flow.</td>
</tr>
</tbody>
</table>

The initial decision should be updated with the new QA in the phase diagram. The flow should resize to the size of the elements inside.  
Pass

<table>
<thead>
<tr>
<th>Step</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Open a phase diagram.</td>
</tr>
<tr>
<td>2.</td>
<td>Add a flow.</td>
</tr>
<tr>
<td>3.</td>
<td>Open the flow diagram.</td>
</tr>
<tr>
<td>4.</td>
<td>Add several design decisions.</td>
</tr>
<tr>
<td>5.</td>
<td>Go back to the phase diagram.</td>
</tr>
</tbody>
</table>

The design decisions should also show in the phase diagram flow element.  
Pass

<table>
<thead>
<tr>
<th>Step</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Create a flow without the project and phase editor.</td>
</tr>
</tbody>
</table>

The behavior should be the same as without the DF Octo+.  
Pass

<table>
<thead>
<tr>
<th>Step</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Open a phase diagram.</td>
</tr>
<tr>
<td>2.</td>
<td>Add a flow.</td>
</tr>
<tr>
<td>3.</td>
<td>Open the flow diagram.</td>
</tr>
<tr>
<td>4.</td>
<td>Add several design decisions.</td>
</tr>
<tr>
<td>5.</td>
<td>Go back to the phase diagram.</td>
</tr>
<tr>
<td>6.</td>
<td>Go back to the flow diagram.</td>
</tr>
<tr>
<td>7.</td>
<td>Delete one of the design decisions.</td>
</tr>
</tbody>
</table>

The design decisions should also be deleted in the phase diagram flow element.  
Pass

<table>
<thead>
<tr>
<th>Step</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Delete the folder repository of one of the flows.</td>
</tr>
</tbody>
</table>

The DF should warn the user when...  
Pass
<table>
<thead>
<tr>
<th>Model integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>
| 1 | Delete the model file of an integrated model and execute the model integration operations. | 1. Open a view diagram.  
2. Add a model element.  
3. Specify the model in the hard drive.  
4. Attach the model with the model integration.  
5. Delete the model file.  
6. Execute one of the model integration operations. | The DF should warn the user that the model file does not exist. | Pass |
| 2 | Place the model file inside the model repository and execute the model integration operations from the command line. | Self explained | The integration operation should execute successfully. | Pass |
| 3 | Place the model file outside the model repository and execute the model integration operations from the command line. | Self explained | The integration operation should execute successfully. | Pass |
| 4 | Try to integrate a model with a format that is not supported. | 1. Open a view diagram.  
2. Add a model element.  
3. Add a file with an unknown format. | No integration should be enabled. | Pass |
<p>| 5 | Integrate a model with a high number of blocks and parameters. | Self explained | The integration operations should execute successfully. | Pass |
| 6 | Execute the command line model intergr-. | Self explained | The DF should warn the user with | Pass |</p>
<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Actions</th>
<th>Expected Result</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Run validation in the View editor.</td>
<td>1. Open a view diagram. 2. Add parameter dependencies between parameters in different flows. 3. Run the validation.</td>
<td>The validation should execute successfully.</td>
<td>Pass</td>
</tr>
<tr>
<td>2</td>
<td>Create a standalone flow diagram.</td>
<td>1. Create a flow without the project and phase editor.</td>
<td>The behavior should be the same as without the DF Octo+.</td>
<td>Pass</td>
</tr>
<tr>
<td>3</td>
<td>Create a new parameter in the View editor and add a new type of unit.</td>
<td>1. Open a view diagram. 2. Add a parameter with a new type of Unit.</td>
<td>The Unit should be added to the core domain knowledge.</td>
<td>Pass</td>
</tr>
<tr>
<td>4</td>
<td>Open the exploration of a parameter and explore to a dependency defined in another flow.</td>
<td>1. Open a view diagram. 2. Add a parameter dependency from a parameters x to a Parameter y in different flows. 3. Open the exploration from Parameter x. 4. Explore towards parameter y.</td>
<td>The exploration should be shown correctly.</td>
<td>Pass</td>
</tr>
<tr>
<td>5</td>
<td>Open the exploration of a parameter and explore to a dependency defined in another flow.</td>
<td>1. Open a view diagram. 2. Add a parameter dependency from a Parameter x to a Parameter y in different flows. 3. Open the exploration from Parameter y and explore towards Parameter x.</td>
<td>The exploration should be shown correctly.</td>
<td>Pass</td>
</tr>
</tbody>
</table>

DF core and Octo+ extensions integration
6.2 Validation
To validate the product, a number of scenarios were established with the stakeholders. These scenarios validate the most important requirements of the product. The scenarios were designed to represent the usage of the product in real situations. For confidentiality reasons only an overview of these scenarios is provided in this section.

6.2.1 Scenario 1: Project, phases and multiple flows modeling of one of the current printer design projects
In one of the current printer design projects, the different phases and flows were modeled in the DF Octo+ by ESI domain experts. For that modeling the architecture and design documentation of that project was studied, and based on those documents actual information on phases, flows, blocks and parameters was added to the DF.

The model was built in the following way:
- The phases of the design process were defined as project phases.
- The aspects of the printer, such as productivity and layout were defined in the DF Octo+ as flows.
- In each flow an architectural document was used to determine the different blocks, parameters and dependencies. The blocks were very different in nature, sometimes they represented functional requirements and sometimes they represented parts of the printer.
- A number of real models were attached to the DF model and the model integration was executed.

This model was later confronted with some architects and their reactions were evaluated. In general, it was noted that the DF Octo+ is better suited to model the design of high-speed production printers and to provide rational about it than the DF Core. Based on this experience some of the stakeholders agreed on performing a live evaluation of the DF Octo+.

6.2.2 Scenario 2: Model integration of some company specific models
Two company specific models were tested for the model integration, a model used to detail key design parameters of the printers and a productivity model to perform calculations of the printer productivity. The parameter model is defined in XML. As the DF Octo+ already provides a generic XML model provider, the model was easily integrated with the DF. The productivity model is an Excel model. As the DF Octo+ offers a provider for Excel models, the productivity model was also easily integrated.

By integrating these models, the model integration was validated against two models used in the design process. The fact that the two models were easily integrated provides an indication of the little effort needed to integrate other models used in the design process in the DF.
7 Project Management

Abstract – In this chapter, an insight into the planning and execution of this project is offered. It explains the way of working, the planning and the risks. It also presents a retrospective on the project with the strengths and possible improvements of the project management.

7.1 Way of working
This project was managed using an iterative agile approach. The project was organized in iterations of between one month and one month and half. The first iterations consisted mainly of technology learning and problem analysis; the mid iterations consisted mainly of product design and development. The last iterations consisted mainly of report writing and product refinement.

To control the process, provide mentoring, and give advice a monthly steering group meeting was held. The steering group was composed of the university supervisor and the ESI supervisors. During this meeting the main aspects of the project were discussed and the general risks evaluated. Additionally, every three months the same group performed a performance evaluation that served as a project assessment.

Besides the monthly steering meetings a weekly meeting was held with the ESI supervisors during the whole project. The role of these meetings was to discuss the work, brainstorm on design alternatives and define design and solutions. Also a similar type of meeting was held with one of the main stakeholders from the printer manufacturing company on a weekly or bi-weekly basis.

7.2 Project Planning and Scheduling
After the first two iterations a high level schedule of the product deliverables was established. This schedule was later adapted as the project progressed. The last version of the schedule can be seen in Figure 43.

![Figure 43: High level schedule](image-url)
It is important to notice that from the original planned deliverables the Relationship Exploration (RELO) was only developed in the alpha version. Later this deliverable was reallocated to another member of the DF development team.

As it was already mentioned the work was divided in iterations. At the beginning of the iteration, and according to the goal of the iteration, a number of tasks were devised and the effort for each task estimated. During the iteration, the task effort was reevaluated to check if the effort required was larger or smaller than the predicted. This information was tracked using an excel template. An example for one of the iterations can be seen in Figure 44.

![Figure 44: Iteration effort estimation and tracking](image)

The example corresponds to the first half of the fifth iteration. The example shows that it took four days more than planned to complete the twenty one story points and that the effort for the last two tasks was larger than the estimated (marked in red).

The story points’ method, typical of Scrum processes, was very useful to predict the amount of work that was feasible to allocate in each iteration. After a couple of iterations it was clear that it was possible to achieve between 25 and 30 story points per iteration.

## 7.3 Risks

The risks of the project were evaluated regularly. The main risks identified can be seen in Table 8.

<table>
<thead>
<tr>
<th>ID</th>
<th>Risk</th>
<th>Probability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Product concurrence between MoBasE and DF</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>Lack of domain knowledge</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>Project scope is too broad</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Lack of commitment in stakeholders</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>Report delayed compared to product</td>
<td>High</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Table 8: Main risks**

For these risks some mitigation strategies were developed. These strategies can be seen in Table 9.

<table>
<thead>
<tr>
<th>ID</th>
<th>Risk</th>
<th>Mitigation</th>
<th>Mitigation effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Product concurrence between MoBasE and DF</td>
<td>Coordinate with MoBasE product owner</td>
<td>MoBaE and DF were not competing products</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Integrate the DF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mitigation strategy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Lack of domain knowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read documentation and talk to experts</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ask to supervisors on the decisions that require a big knowledge of the domain</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decisions taken with more information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Project scope is too broad</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Negotiate with supervisors on the scope</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Limitation of the scope</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Lack of commitment in stakeholders</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ask to supervisor to introduce the project in meetings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Plan meeting with stakeholders well in advance</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stakeholders are more committed with the project and understand it better.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Report delayed compared to product</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Advance in parallel. Have different versions of the report be checked by the supervisors.</td>
<td>Quality of the report is improved, risk of finish the product but not the report is reduced.</td>
<td></td>
</tr>
</tbody>
</table>
8 Conclusions

Abstract – This chapter describes the results of the project and provides an outlook into interesting future developments of the DF.

8.1 Results
The main goal of this project was to adapt the DF to better represent the design process of high-speed production printers. To achieve this goal, some extensions to the DF were developed and are now available as plug-ins for Eclipse. These new plug-ins enable the printer manufacturing company to:

- Extend the DF to be used in larger and more complex projects than it was before through the project, phases and multiple flows.
- Make an important step towards the federation of interconnected models and the virtual printer through the model integration.
- Provide more powerful visualizations and explanations through the relationship exploration.

These extensions were designed to be installed on top of the DF Core, enabling ESI to use the DF Core in other companies which was an important requirement.

8.1.1 Project, phases and multiple flows
A new editor called Project editor was introduced. The Project editor enables to collect all the project information in a centralized place. The Project editor is used to edit a project and define phases. The order of the phases can also be established in this editor. When a phase has a previous phase, the flow structure of the previous phase is copied to the phase and a link between the flows is created. These links enable to import information from the previous phase, as shown in Figure 46. The project, phases and multiple flows is explained with the DF design case. The Project editor can be seen in Figure 45, with the design phases: Multiform and Octo+.

![Figure 45: Project editor](image)

A new editor called Phase editor was also introduced, as shown in Figure 46 and Figure 47. This new editor can be opened by selecting a phase in the Project editor.
The Phase editor enables the teams and designers to provide rationale for the different aspects or subsystems of the design, having a flow for each aspect or subsystem. It also enables to connect them through the flow dependencies if the parameters or blocks that characterize these flows have relations between themselves.

The View editor was extended to enable the creation of dependencies between designs in different flows.

In the DF example shown in Figure 47, the Multiple Flows design has dependencies to the RELO design.

The Flow editor was also extended and the flows created in the Phase editor can be opened with the extended version of it. Each Flow has its own model and diagram file. Typically, a team or designer will work in this editor to model their part of the system design.
Once a flow dependency is created, it is also possible to evolve it in the future. If one of the system dependencies changed, then the user will be informed about it and can migrate it to the correct version.
8.1.2 Model integration

A new mechanism called model integration was introduced. The model integration enables the printer manufacturing company to create model providers that “understand” the model formalism and can be easily integrated in the DF. The model integration was created using a plug-in architecture so that it is very easy to add new providers for new types of models.

To add a new model provider one just needs to implement one interface and create an Eclipse plug-in project to deploy it to the DF, which can be done in a few clicks. The DF Octo+ comes with predefined model providers for XML and Excel models.

To integrate a model the user needs to select that model in the DF and attach it by selecting the mapping from the DF to the external model.

![Figure 50: Attaching model using provider](image)

![Figure 51: Map external model to DF](image)

![Figure 52: Parameters copied and mapped in DF](image)

It is possible, to import, to export and to run analysis on the model from within the DF.
The model integration can also be invoked from a command line interface to integrate it into a build process or some batch environment.

![Figure 53: Command line model integration](image1)

### 8.1.3 Relationship exploration

A new mechanism called relationship exploration was introduced. Using the relationship exploration it is possible to explore the system (using the relation between the elements) from any part of it. It is also possible to use this mechanism to create specific views of the system from the perspective of a particular stakeholder. For example, the Figure below shows an exploration of a camera design parameters related to the power consumption for a stakeholder interested in it.

![Figure 54: Relationship exploration](image2)

### 8.2 Reflection

This section presents a discussion of the results of this project. This discussion is performed not only from the point of view of the artifacts but also from the point of view of the validation of the DF concept and design.
8.2.1 Applicability of the solution

As a part of this project several extensions were devised to model the printer design process with the DF. These extensions were discovered through an analysis of the printer manufacturing company design process and were not evaluated in the context of other companies. However, these extensions appear to be generic to several design processes.

It is expected that in other companies of the high-tech sector the projects can also be modeled under phases and multiple flows. It is also expected that they produce multiple models in their design process and want these models to be easily integrated in the DF.

In consequence, it is possible that many of these extensions become part of the DF Core, maybe after some changes in the concept. However, this is just an assumption and the extensions need to be tested in several other contexts to validate this assumption.

8.2.2 DF Meta-model extensibility layer

The DF Core meta-model contains an important set of abstract elements that intend to provide extensibility to the DF, however the extensibility provided is very limited and could not be used in this project to create the extensions.

There is a fundamental problem to use this extensibility; it is only usable from the EMF Editor and thanks to an EMF mechanism that searches in the meta-model for extended classes. From the GMF editors point-of-view this extensibility is useless.

The Flow and View editors are totally tied to the implementations and do not know about these abstract elements. If you add a new class that extends one of the abstract classes the Flow and View editors would not work. In this project, the Flow and View editors needed to be reused so this mechanism did not help.

It is also arguable if this extensibility layer could help in other cases; the DF concrete classes are already very generic by definition and most of the DF value and behavior
is in the editors. Therefore, it is even questionable if the added value obtained from this extensibility layer justifies the design complexity that it adds.

### 8.2.3 DF Meta-model

During this project it became quite evident that an important disadvantage in the design of the DF Meta-model is the lack of a common base class for all the editable elements. It is a common practice in many standardized meta-models such as UML [16] and BPMN [17].

![Diagram: Top abstractions in UML meta-model](image)

Particularly, in the DF core meta-model it would be very useful to have a base class for all the named elements that contains the name of the element and a base class for all the relations that contains the source and target elements. There would be many operations on the meta-model that would be simplified with these inclusions. For example, the search of an element in the meta-model, that is a very common operation.

### 8.2.4 EMF and GMF Model Driven Development

The DF is based on a Model Driven Development approach. Models are created in EMF and GMF and Java classes are obtained from them. It is unquestionable that it is a good idea to raise the level of abstraction and perform Model Driven Development. However, it is questionable if the models defined in EMF and GMF are not too simple. The reason is that too much development is done in the generated classes in the case of GMF and EMF.

Particularly in the case of GMF, the editors created from the models are in many cases too simple or too far from the desired functionality. The developer has to learn the formalism to construct the models and also the detailed implementation of the generated code and the GMF libraries to be able to modify the editors. Therefore, it is debatable if the effort of using GMF MDD is not even higher than the effort of just developing the editors in code directly.
8.3 Future Work

The DF Octo+ introduced some core features for the application of the DF in the printer manufacturing company. However, there are still a number of features that could improve the DF and help even more the designers. Following are some of these features:

- A live collaboration environment. Currently the DF models are used through SVN to provide the multi user environment. However, by using SVN some of the DF features can only be used off-line and the conflict resolution can become very complicated in some cases. There is a very promising project called CDO Dawn to port GMF editors to be used in a live collaborative environment. A feasibility study was performed and with very little changes the Flow and View editor were used with Dawn. With Dawn the DF could offer the following:
  - Live notifications. When a designer change something another designer could be automatically notified. For example for the validation.
  - Easier conflict resolution. As the editing is online the conflicts become easier to resolve.

- What-if analysis. The DF can be used to perform analysis by changing some of the parameters in the models and establish the impact on the rest of the system.

- Drag and drop of blocks and parameters as explained in the requirements.

- More flexible views. Currently the view elements in the DF behave like the blocks, they are containers. However, it would be more convenient if there would be a unique system design, and the views would work only as a visualization mechanism.
**Glossary**

<table>
<thead>
<tr>
<th>AM</th>
<th>Architecture Model. Is a tool to model the architecture of systems.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMF</td>
<td>Eclipse Modeling Framework. It is an Eclipse based framework to create meta-models by using a model driven approach.</td>
</tr>
<tr>
<td>Eclipse</td>
<td>It is a platform to create software applications.</td>
</tr>
<tr>
<td>EditPart</td>
<td>It is the controller in the GEF framework.</td>
</tr>
<tr>
<td>EditPolicy</td>
<td>It is a class used by the EditParts to answer to the user requests in GEF.</td>
</tr>
<tr>
<td>Feature</td>
<td>It is a unit of deployment in the Eclipse platform.</td>
</tr>
<tr>
<td>GEF</td>
<td>Graphical Editing Framework. It is an Eclipse based framework to create editors based on the Model-View-Controller pattern.</td>
</tr>
<tr>
<td>GMF</td>
<td>Graphical Modeling Framework. It is an Eclipse based framework to create editors by using a model driven approach. It uses EMF and GEF.</td>
</tr>
<tr>
<td>Jar</td>
<td>It is a unit of deployment in the Java platform.</td>
</tr>
<tr>
<td>Meta-model</td>
<td>A meta-model defines the language and processes from which to form a model.</td>
</tr>
<tr>
<td>Semantic Element</td>
<td>An alternative name for meta-model element used in GMF.</td>
</tr>
<tr>
<td>RELO</td>
<td>Relationship exploration</td>
</tr>
<tr>
<td>SVN</td>
<td>Subversion. A configuration management system.</td>
</tr>
</tbody>
</table>
Bibliography


[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software.: Addison-Wesley Professional, 1994.


About the Author

Martin Palatnik received his Computer Engineer degree in January 2010 from the Catholic University of Uruguay. His thesis was conducted in the Business Process Management field and is entitled “Assigning Work Items more efficiently by detecting participants’ busyness and predicting workload using Business Intelligence tools”. As a part of his Engineering degree he took a year in the University of Leeds where he later performed an internship in the Computer Vision group.

Between 2010 and 2012 he worked at the Stan Ackermans Institute of Eindhoven University of Technology, where he completed the Software Technology program.

Additionally, he worked as a Software Engineer in the development of a lending system for the main consumer lending company of Uruguay.
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.