Océ Datapath Editor

User Requirements Document

Department of Computer Science, Eindhoven University of Technology

Eindhoven, 07-03-2010

Advisor
Ion Barosan, HG 5.92
Senior Management
Lou Somers, HG 5.36
Customer
Egbert Teeselink, Océ

ODE Group
Pieter Bootsma, 0619781
Sebastiaan Candel, 0619708
Gijs van Enckevort, 0608394
Vincent Huisman, 0609075
Sam Mousa, 0567800
Gerard de Ruig, 0555781
Linda Spaninks, 0634433
Lourens Visscher, 0554498
Quality Assurance Manager
Florian van der Wielen, 0587140
Project Manager
John Servaes, 0661508

Abstract

This document contains the user requirements for the ODE (Océ Datapath Editor) application. This software product is part of the Software Engineering Project (2IP35) at the Eindhoven University of Technology.

These user requirements were established according to requests formulated by Group ODE taking into account the wishes of our customer, Egbert Teeselink, on behalf of Océ.

This document complies with the specifications for a User Requirements Document (URD) by the software engineering standards, as set by the European Space Agency [2].

During the project the group will create a tool in which Océ's software engineers can model a datapath.

Document Status Sheet

Document Name	User Requirements Document
Document Identification	ODE/DOC/URD/1.0.437
Document Author(s)	Sebastiaan Candel, Gijs van Enckevort and Gerard de Ruig
Document Status	Externally accepted

Document history

Version	Date	Author(s)	Summary
0.1	11-02-2010	GE, GR and SC	First internally reviewed version
0.2	22-02-2010	GE, GR and SC	Second internally reviewed version
0.3	01-03-2010	GE, GR and SC	First internally accepted version
1.0	04-03-2010	GE, GR and SC	First externally accepted version

Document dependencies

This document has no dependencies.

List of names and their abbreviatons

Abbreviaton	Full Name
PB	Pieter Bootsma
SC	Sebastiaan Candel
GE	Gijs van Enckevort
VH	Vincent Huisman
SM	Sam Mousa
GR	Gerard de Ruig
LS	Linda Spaninks
LV	Lourens Visscher

Document change record

Document Title	User Requirements Document
Document Identification	ODE/DOC/URD/1.0.437

Changes

Section	Subsection	Changes
*	*	Revised according to QAM and customer feedback
1	1	Final changes according to customer feedback
1	2	Final changes according to customer feedback
2	2.1	Final changes according to customer feedback
2	6	Final changes according to customer feedback
3	1.2	Final changes according to customer feedback
3	3	Final changes according to customer feedback

Contents

Chapter 1

Introduction

1.1 Purpose

The User Requirements Document (URD) contains the requirements for the Océ Datapath Editor (ODE). The document conforms to the ESA document standard [2]. These requirements are a negotiated agreement between Océ and the project team. All of the listed requirements, and only these, will be implemented according to their priorities. Any changes to these requirements require the consent of both parties.

1.2 Scope

The ODE Group will provide Océ with an editor where the user can model a so called datapath. The editor will aid the process of choosing the correct hardware, required for the tasks a printer or copier should be able to execute. The ODE will provide a visual environment in which the user can create and change these models and export them for analysis. The editor will also get feedback from the analysis tools, e.g. highlighting specific items in a model.

The function of the product at Océ will be to provide a correctly structured file for use in Océ's own analysis tool chain. See Figure 1.1 for a visual representation. Because it is irrelevant of which actions the analysis and verification tools consist, it is depicted as an abstract "Analysis" object.

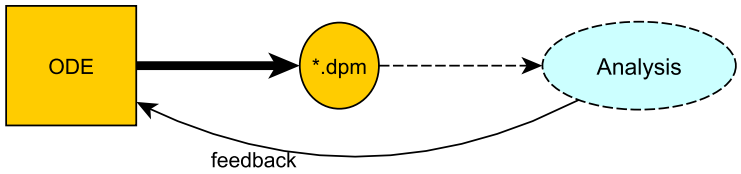

Figure 1.1: Position of the product in the entire software line.

1.3 Definitions, acronyms and abbreviations

This section contains a list of definitions for used acronyms and abbreviations.

GUI	Graphical User Interface
Océ	Océ-Technologies B.V.
ODE	Océ Datapath Editor
URD	User Requirements Document
ESA	European Space Agency
SRD	Software Requirements Document
SR	Software Requirements Phase

1.4 List of references

[1] Steve Ash. Moscow prioritisation briefing paper. April 2007.

[2] ESA. Software engineering standards. *ESA PSS-05-0*, Issue 2, February 1991.

[3] ODE. Software requirements document. March 2010.

1.5 Overview

The remainder of this document is divided into two chapters, General description and Specific requirements.

The *General description* gives a clear and concise description of the ODE. It consists of a short view on the product, a short summary of the general capabilities of the product and a more in-depth view of the main parts of the product: the model editor and the repository.

The *Specific requirements* are subdivided in capability requirements, describing what the system should do, and constraint requirements, describing how the requirements should be implemented.

Chapter 2

General description

2.1 Product perspective

The ODE will provide a graphical interface for designing models representing datapaths, which can be exported and analysed by other software. The idea of using software to design and analyse datapaths in order to make the engineering of new printers, scanners and copiers easier and faster has been exploited by Océ for a while, but they have not been successful at creating a single tool that can generically model any datapath. The analysis and verification tools are still in development and the exact representation of data used by verification tools has yet to be determined. Defining these standards and using them is beyond the scope of this project. The ODE should provide the user with a visual way of modelling datapaths and it should also allow the user to export this model to a file for analysis and verification. Besides being easy to use, it is also important that future developers can easily expand its functionality and adapt the input or output format.

2.2 General capabilities

The ODE will consist of two parts, the model editor and a repository. The model editor will provide a visual way to model datapaths consisting of an application part, a platform and a mapping that maps the application onto the platform. The repository will allow the user to add and edit specific tasks, algorithms and hardware types in text. Definitions for tasks, algorithms and hardware types are given in a later stage. Both parts will be described separately below. Some definitions or terms used are listed in the following table.

Application node	A generic item in the application.
Application edge	A generic directed edge between nodes in the application.
Platform node	A generic item in the platform.
Platform edge	A generic undirected edge between nodes in the platform.
Link	A connection between an application node and a platform node.
Mapping	The collection of all links.
Project	A project consists of a set of project properties, one application, one platform and one mapping between the application and platform.
Repository item	An item that can be edited in the repository, either a hardware type, task or algorithm.

2.2.1 Model editor

This part of the ODE provides a GUI in which the user can model an application, a platform and a mapping that maps the application onto the platform. The entire structure of the model has yet to be defined and is prone to change after this project has been finished. However, some details about the structure of various nodes and edges are known and unlikely to change during the course of this project. These details are described in the subsections below. They have the following concepts:

Application

The application consists of a set of application nodes and edges between these nodes. The edges in the application are directed from node to node.

There will at least be a type of node called *step*, some kind of *start node* and some kind of *end node*. A *task* can be assigned to a step, describing the properties of that specific step (e.g. *scan*, describing the speed and size of the scan). These tasks are stored in the repository.

A typical application will contain 8 nodes. Applications will generally contain at most 20 nodes. For example, an application could be visually represented as in Figure 2.1.

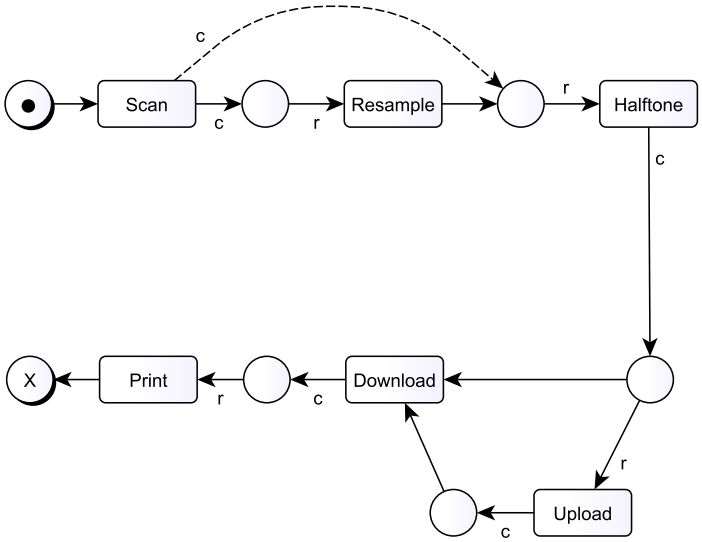

Figure 2.1: An example of an application.

Platform

A platform is an undirected graph consisting of platform nodes and edges between these nodes. These nodes represent the hardware elements that the device will use to physically execute the tasks as defined in the application.

There will at least be a type of node called *resource block*. A specific *resource* can be assigned to a resource block, describing the properties of that specific resource (e.g. *512 MB RAM*, describing the size of the memory). These resources are stored in the repository. For example, a platform could be visually represented as in Figure 2.2.

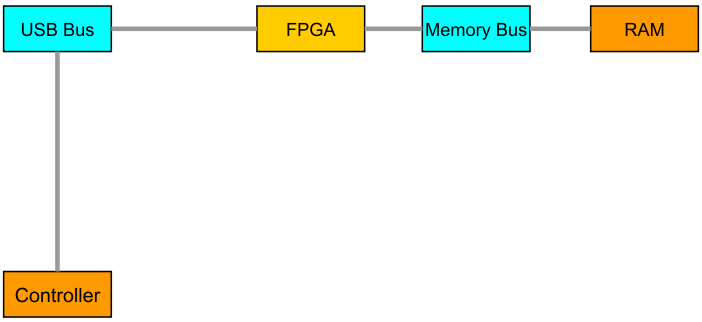

Figure 2.2: An example of a platform.

Mapping

The mapping maps an application onto the platform used to execute it. Each link from an application node to a platform node can have an associated *algorithm* that is stored within the repository. The algorithm specifies how efficient a task can be executed on this piece of hardware.

2.2.2 Repository

The repository is included to allow users to add, edit and remove defined algorithms, resources and tasks. Within each entry it is possible to add comments about the algorithm, resource or task in question.

2.2.3 Maintainability and extensibility

Because the export format and the input language are prone to modifications, it is important that future developers can easily modify the code to comply with new standards. Besides changes in the export format, it is also probable that the functionality of the product will be expanded. Therefore the quality standards for the source code of the ODE are high.

2.3 General constraints

The ODE is not designed to check the semantics of a given model. However, it may check for basic validity of models created with the ODE and provide constraints within its GUI to prevent some basic syntactical errors. Details of these constraints are given in the SRD [3].

2.4 User characteristics

There are two types of users that interact with the final product. The first group is a small group of users that use the actual program to model datapaths. These users have in-depth knowledge of what can be modeled with the product.

The second group of users does not use the product itself, but uses its code to add more features to the editor. These users are also very important since the export format and the product's required functionality are prone to change. Obviously, this results in high demands to the quality of the code and architecture.

2.5 Operational environment

The product will run on computer systems running either Windows XP or Windows 7 as primary operating system. It will only be used by Océ's in-house software engineers.

2.6 Assumptions & dependencies

In order for us to be able to provide a good product, we expect the customer to comply to several requirements. These requirements have been formulated in cooperation with the customer.

- The language to be used in the models is frozen before the end of the SR.
- The language, when frozen, is clear and concise.

- The semantics of the model are provided by the customer to the extent that matters to the editor.
- The semantics of the model are clear and concise and do not receive significant changes to the extent that matters to the editor.
- The number of constraints inferred by the customer is small.

Chapter 3

Specific requirements

This chapter covers the capability requirements and constraint requirements. Each requirement consists of a unique identifier, followed by a short description and a priority. We will use the following priorities, ordered from highest to lowest priority, as defined in [1]:

***Must have*:** these requirements are essential for the product.

***Should have*:** these requirements are not critical for the product to work, but are nearly as important as the Must haves, meaning they must be implemented if at all possible.

***Could have*:** requirements which are not critical to the products success. If they can be implemented with little development costs, they can increase customer satisfaction.

3.1 Capability requirements

UR-C-0 Any application node and platform node can be selected by the user.	*Must have*
UR-C-1 It is possible to select any application node or platform node by means of a method call.	*Must have*
UR-C-2 It is possible to highlight any application node or platform node by means of a method call.	*Must have*
UR-C-3 Other programs can send messages to the product for selecting and highlighting edges and nodes.	*Could have*

3.1.1 Model editor

UR-D-0	*Must have*
The user can create a new project.	
UR-D-1	*Must have*
The user can open an existing project.	
UR-D-2	*Must have*
The user can save an open project.	
UR-D-3	*Must have*
The user can close an open project.	
UR-D-4	*Must have*
The editor automatically saves the current projects frequently to a default directory.	
UR-D-5	*Must have*
The user can undo changes to the model made in the current session.	
UR-D-6	*Should have*
The editor supports alignment of any element (such as application nodes and platform nodes) relative to similar elements in its vicinity.	
UR-D-7	*Must have*
When nodes are moved, edges move along.	
UR-D-8	*Must have*
When two nodes are copied which have an edge between them, this edge is copied along.	
UR-D-9	*Must have*
It is possible to select multiple objects (such as nodes, edges or links) at the same time.	
UR-D-10	*Must have*
It is possible to add a number of objects to a selection.	
UR-D-11	*Could have*
It is possible to export the application or platform in a BMP image format to the clipboard.	
UR-D-12	*Could have*
It is possible to export the application or platform to a EMF vector image format to the clipboard.	
UR-D-13	*Should have*
It is possible to save the application or platform to a BMP image file.	
UR-D-14	*Could have*
It is possible to save the application or platform to a EMF vector image file.	

Application

UR-DA-0	*Must have*
The user can add an application node to the application.	
UR-DA-1	*Must have*
The user can add an edge to the application.	
UR-DA-2	*Must have*
The user can connect edges to application nodes.	
UR-DA-3	*Must have*
The user can assign a task to a step.	
UR-DA-4	*Must have*
The user can assign a name to a starting node.	
UR-DA-5	*Should have*
The user can change the task assigned to a step.	
UR-DA-6	*Must have*
The user can move application nodes and edges within the application.	
UR-DA-7	*Must have*
The user can remove application nodes and edges from the application.	
UR-DA-8	*Must have*
The user can cut, copy and paste (parts of) the application to an (other) application.	

Platform

UR-DP-0	*Must have*
The user can add a platform node to the platform.	
UR-DP-1	*Must have*
The user can assign a resource to a resource block.	
UR-DP-2	*Should have*
The user can change the resource assigned to a resource block.	
UR-DP-3	*Must have*
The user can connect platform nodes.	
UR-DP-4	*Must have*
The user can move platform nodes within the platform.	
UR-DP-5	*Must have*
The user can remove platform nodes from the platform.	
UR-DP-6	*Must have*
The user can cut, copy and paste (parts of) the platform to a(n) (other) platform.	

Mapping

UR-DM-0	*Must have*
The user can create a link between an application node and a platform node.	
UR-DM-1	*Must have*
The user can assign an algorithm to a link.	
UR-DM-2	*Must have*
The user can change the algorithm assigned to a link.	
UR-DM-3	*Should have*
The user can connect the starting point of a link to a different application node.	
UR-DM-4	*Should have*
The user can connect the endpoint of a link to a different platform node.	
UR-DM-5	*Must have*
The user can remove a link.	
UR-DM-6	*Could have*
The editor will assign an algorithm to a link if there is exactly one algorithm available for it.	

3.1.2 Repository

UR-R-0	*Must have*
The user can add an algorithm, resource and task to the repository.	
UR-R-1	*Could have*
The user can add an algorithm, resource and task to the repository using a fancy wizard.	
UR-R-2	*Must have*
The user can open an existing algorithm, resource and task in the repository.	
UR-R-3	*Must have*
The user can remove an existing algorithm, resource and task from the repository.	
UR-R-4	*Must have*
The user can modify an existing algorithm, resource and task in the repository.	
UR-R-5	*Must have*
The user can close an open algorithm, resource and task in the repository.	
UR-R-6	*Must have*
The user can undo any changes made in the text editor of the repository.	
UR-R-7	*Could have*
The repository supports basic syntax highlighting.	
UR-R-8	*Must have*
It should be easy to add syntax highlighting to the repository.	

3.2 Constraint requirements

3.2.1 Model editor

UR-D-15	*Should have*
Constraints posed by the modeling language are enforced by the ODE. These constraints are specified in the SRD [3]	

3.3 Non-functional requirements

UR-N-0	*Must have*
Every method and class in the source-code is documented using comments, explaining what it does, and how it does this. These comments have to be documented according to standards enforced by the documentation generation tool that the ODE group chooses to use.	
UR-N-1	*Should have*
At least 80% of the methods are functionally cohesive or sequentially cohesive.	
UR-N-2	*Should have*
Every method and variable in the source-code has a clear and verbose name.	
UR-N-3	*Must have*
The ODE runs on Windows XP and Windows 7.	
UR-N-4	*Must have*
The entire source code complies with the coding standard provided in the SRD[3].	
UR-N-5	*Must have*
There is a unit test for every method or class that is not directly responding to a mouse-click or keystroke.	
UR-N-6	*Should have*
Every method and class is tested successfully.	
UR-N-7	*Must have*
The ODE does not measurably suffer from memory leaks.	