Master Thesis

A Scenario Editor for Design Space Exploration

Sander van Zuidam
December 2009

Supervisor: dr L.J.A.M. Somers
Eindhoven University of Technology
Abstract

The data processing and the digital stages of complex systems, such as printers, are called a data path. During the development of these systems, developers specify various alternatives for data path architectures, consisting of devices that perform the necessary data processing (CPU, FPGA), intermediate storage (RAM, disk) and transferring. The choices made have a big impact on data path performance. Other concerns developers have to keep in mind are e.g. time-to-market and the price of the various parts. There is definite need to support the task of developers by means of software tools that show the consequences of choices made or that even suggest improvements to allow developers to decide which data path architecture alternative is the best one. This way, the idea for a DSE (design space exploration) tool chain was born.

In this report the scenario editor is discussed, the tool in the DSE tool chain to support developers with developing a data path in a structured way with a graphical user interface. With the scenario editor a data path can be designed and presented as a graphical model, which will then be analyzed by other parts of the chain and finally the visualization tool will present the performance results.

The scenario editor is developed with the Y-chart approach in mind. The Y-chart approach allows for a general description of data paths, by defining the data processing units, performing devices and the mapping of these two separately. This way a design space exploration approach can be used on the data path to allow for finding problems (e.g. performance bottlenecks).

While the developed scenario editor is certainly further expandable, for static (fixed order and all parameters determined at analysis time) data paths it is already quite powerful.
# Index

1. Introduction ........................................................................................................ 5  
   1.1 Context .......................................................................................................... 5  
      1.1.1 Embedded Systems Institute (ESI) ...................................................... 5  
      1.1.2 Océ ................................................................................................. 5  
   1.2 Octopus Project: System Adaptability ....................................................... 5  
2. Problem Analysis .............................................................................................. 6  
   2.1 DSE Tool chain .......................................................................................... 6  
   2.2 Y-chart approach ....................................................................................... 7  
   2.3 Data paths .................................................................................................. 8  
   2.4 The assignment ........................................................................................... 8  
3. Tool chain architecture ..................................................................................... 9  
   3.1 Kernel ........................................................................................................ 9  
   3.2 Analysis tool(s) .......................................................................................... 11  
   3.3 Visualization tools ...................................................................................... 11  
4. Requirements .................................................................................................. 13  
   4.1 Input Requirements ................................................................................... 13  
   4.2 Interface requirements .............................................................................. 13  
5. Scenario Editor Architecture ........................................................................... 14  
   5.1 Application and hardware program .......................................................... 15  
      5.1.1 Application design primitives ............................................................ 16  
      5.1.2 Platform design primitives ................................................................ 19  
   5.2 Mapping program ....................................................................................... 20  
      5.2.1 Development environment ................................................................ 21  
      5.2.2 Design primitives ............................................................................. 21  
   5.3 Connecting the two parts ........................................................................... 22  
6. Implementation ............................................................................................... 23  
   6.1 Application and hardware program ......................................................... 23  
   6.2 Mapping program ....................................................................................... 26  
   6.3 Connecting the two parts ........................................................................... 28  
7. Interfaces ......................................................................................................... 29  
   7.1 Within the Scenario Editor ........................................................................ 29  
   7.2 Between Scenario Editor and Kernel ....................................................... 29  
   7.2 Between Kernel and CPN Tools ............................................................... 31  
   7.3 Threading ................................................................................................... 31  
8. Scenario editor files ......................................................................................... 32  
   8.1 File extensions ............................................................................................. 32  
   8.2 File structure ................................................................................................ 33  
   8.3 Options file .................................................................................................. 35  
9. Evaluation ......................................................................................................... 36  
   9.1 Scenario editor ............................................................................................. 36  
   9.2 Eclipse ......................................................................................................... 36  
   9.3 Maintainability ............................................................................................ 37  
10. Conclusions .................................................................................................... 38  
   10.1 Goals .......................................................................................................... 38  
   10.2 Impact ......................................................................................................... 38  
   10.3 Future work / Change cases .................................................................... 38  
      10.3.1 Using the programs for data paths of other products ....................... 38  
      10.3.2 Adding scheduling algorithms ......................................................... 38  
      10.3.3 Showing connections ....................................................................... 39  
      10.3.4 Feedback ......................................................................................... 39
10.3.5 Statistics as parameters .................................................................40
11. Literature & Links .................................................................................41
12. Appendix A: Eclipse GMF Maintenance guide ........................................42
1. Introduction
In this chapter the context for the scenario editor is discussed. The scenario editor is a part of the Design Space Exploration tool chain for the Octopus project. The Octopus project is a cooperation project between the Embedded Systems Institute, four universities and Océ Technologies. The four universities are the Delft University of Technology, the Radboud University Nijmegen, the University of Twente and the Eindhoven University of Technology.

1.1 Context

1.1.1 Embedded Systems Institute (ESI)
ESI [1] is one of the few research institutes in the world that address embedded systems design at the multidisciplinary systems level. ESI distinguishes itself through its positioning between academia and industry and through the strong industrial role in its research projects. It is the explicit goal of ESI to become a center of expertise with international, technological leadership in embedded systems design. The research projects are the means to achieve this goal. To realize this, ESI has a research staff to transform and consolidate the results of the research projects into tangible knowledge and expertise.

1.1.2 Océ
Océ is one of the world’s leading providers of document management and printing systems. The Océ product range includes high speed digital production printers and wide format printing systems for both technical documentation and color display graphics, as well as office printing and copying systems. Océ is also a foremost supplier of document management outsourcing. Many of the Fortune Global 500 companies and leading commercial printers are Océ customers. The company was founded in 1877. With headquarters in Venlo, The Netherlands, Océ is active in approximately 100 countries and employs some 22,000 people worldwide. [2]

1.2 Octopus Project: System Adaptability
The Octopus project[1] is a collaboration project between ESI, four universities and Océ Technologies. The main goal of this project is to improve system adaptability. This project has many research challenges for the creation of these adaptable systems, i.e. designing approaches to analyze the adaptable properties of the system in its environment.

Within the project there are three lines of attention which are all tightly coupled with the development of digital printers:

- **Data Path Design**
  - the digital stages from network or scanner to print head or network with several types of image processing

- **Adaptive Control of the Physical Layer**
  - the control of the printing device itself

- **System Level Reasoning and Design**
  - the overall architecture of the adaptable system
2. Problem Analysis

This section describes the problem analysis of this graduation project. First are the current situation and the approach in mind for the solution described. Then is data path design explained and finally the actual assignment is described.

2.1 DSE Tool chain

Printers are no longer simple machines that receive a digital document and print it on paper, but complex machines that can print on different kinds of paper, scan documents, perform operations on documents, store files or transfer them. The development of these printers is done by researching new hardware techniques to implement and also by data path design to model the operations of the printer to develop. For the latter, it is nice to be able to test various designs.

Because printers get more complex and the range of hardware parts to choose from gets wider, the need for a program chain has risen. With this program chain printer data paths can be used to develop and analyze the results of using hardware parts for different use cases.

This program chain focuses on Design Space Exploration (DSE) to determine the best solution. DSE checks all possible values (within certain constraints to handle size and complexity) of the data path parameters and the best solution is returned as a result. This DSE approach is used to determine the performance metrics of a printer data path modeled in the chain.

There already exist tools to analyze the performance of embedded systems. These mostly concern the software part of embedded systems[11], but there are also solutions which allow the user to both specify the software and the hardware part, as the CARAT tool[9]. Analysis methods have their limitations, such as only analyzing static environments. The DSE tool chain only analyzes the data paths, not the software. It uses various analysis methods to cover the weak points that would exist if only one analysis method is used[10].

The main advantage of the DSE tool chain is the ability to abstract from the complex languages that existing analysis and modeling tools use by offering a low profile tool.
2.2 Y-chart approach

The idea behind the DSE tool chain and the scenario editor is the Y-chart approach for Design Space Exploration[15]. This approach defines an application (the tasks to perform) and a platform (resource definition). These two are combined in the mapping, that defines which tasks run on which resources. After analysis by analysis tools has been performed on the mapping, results will emerge. The results may lead to improvements in the application, platform or mapping (see figure 2.2.1). By using this approach for the DSE tool chain, the chain will be much more flexible and it is easily adapted for other uses than printer data path development.

![Figure 2.2.1: Y-chart approach, dotted lines are feedback lines](image)

The application, platform and mapping are defined separately, but have their dependencies. The most obvious dependency is in the mapping. No mapping can be created without the application and platform (which can also be seen in the figure). Also when creating the application, it should at least be known what is possible with the platform that will be available. This is a minor dependency and should not influence the developer in defining the application and platform separately.
2.3 Data paths

A data path is defined as a collection of functional units that perform data processing operations. These functional units have parameters and an order in which they have to be performed. This order does not necessarily have to be sequential. An example of a data path can be found in video editing. When editing a video, first the computer has to load the video, then add subtitles and then convert the video to a different format. Finally, the video is stored again. These steps (load, add subtitle, convert and store) are the functional units of the data path. Parameters can be things like the file length or locations.

Functional units use resources to perform their operations. In the video editing example, the resources can be the processor, the hard disk or the graphics card. Certain functional units can use multiple resources and their functions (e.g., throughput) can depend on the chosen resource (resource sharing). For this reason, a data path also contains the information about which functional unit uses which resource.

Data path usage also plays an important role in printer development. As said before, today's printers are complex machines. An example of a printer data path will be given; it will be explained with the Y-chart approach in mind.

In the application part of a data path, the actions (data processing operations) that the printer will take are described. The actions can describe a complex use case, such as scanning a page and then fax and print that page concurrently for efficiency. It might even print another document that has been sent by another user while scanning. If one wants to describe this use case in a formal way, where it is clear which actions can follow which and what priorities there are, this needs to be done in a clear matter. This is one of the challenges when designing a data path.

Another challenge comes from the platform (resources). Some application actions might use the same resources (resource sharing) and can therefore not be executed at the same time, or maybe they can, but in that case, they will be slower. And if there are multiple parts of the platform on which an action can run, which one is best? Would it run faster if that piece of the platform was upgraded? These are all questions that make data path design nontrivial.

Because this is nontrivial, a way to automate the performance analysis of data path architecture has been made. The DSE tool chain contains a number of analysis tools (see section 3.2). To help developers in understanding the analysis results, the results are presented by a visualization tool (see section 3.3).

2.4 The assignment

The goal of the graduation assignment is to develop the scenario editor, the first program of the DSE tool chain that will be used to specify and analyze data paths for printers. In this program, the application (use cases) and platform (hardware) of the printer data path will be defined, and the mapping between these two. In these three steps, all information will be given that is necessary for the analysis tools to perform. The application will contain the actions of the printer data path, the hardware (platform) will contain the technical information of the printer data path, and the mapping will combine these two and makes it possible for the analysis tools to i.e. determine how long the application will take.

In the scenario editor, the developers will be able to make printer data paths in a simple, user-friendly way. They will not have to learn any of the analysis tool languages to be able to operate the scenario editor.
3. Tool chain architecture

In this chapter the architecture of the DSE tool chain is discussed. As the purpose of this graduation project was to develop the scenario editor, this chapter will only discuss the other three parts of the tool chain, namely the kernel, analysis tool(s) and the visualization tool. The parts of the chain are discussed in this order.

The DSE tool chain consists of four parts (see figure 3.1): the scenario editor, in which the user can specify the application, platform and mapping of these two for the data path; the kernel, which does pre-analysis and transforms information from the scenario editor to a general format; the analysis tool, which performs the analysis of the printer specified by the user in the scenario editor and finally the visualization tool that visualizes the analysis results such that the user can read the results in a understandable way. A future improvement is to allow feedback from the results of the analysis tool or visualization to be passed to the editor. One example of feedback is that the analysis tool detects a performance bottleneck which it reports to the editor.

![Figure 3.1: DSE tool chain architecture](image)

3.1 Kernel

The kernel is the heart of the DSE tool chain[6]. All tools of the chain are connected to the kernel. The kernel has no user interface and can only be invoked by the scenario editor. The analysis tools and visualization tool are invoked by the kernel. The kernel is written in Java and is embedded into the scenario editor, running in a separate thread and invoked on demand.

Inside the kernel, the information obtained from the editor is first stored in the general specification model[4] and pre-analysis is performed by forming a partial order from this specification (this is explained in more detail later in this section). This partial order is unfolded and presented to the user by POVis (see section 3.3). The general specification is transformed to the format the user-chosen analysis tool needs for analysis.

A general class diagram is used to represent the general specification model (see figure 3.1.1).
The general specification model[4], in which the kernel stores the data obtained from the scenario editor, specifies a UseCase which has Events (steps in the use case) that have an Ordering and are performing a certain job (GenJob). The split of use case tasks into Events is necessary to allow for a more detailed ordering in which situations like parallelism (tasks that overlap), concurrency and pipelining can be described. This is done because Events are instantaneous and tasks are not. Each Event can either Claim or Release a certain amount (amt in the model) of resources (represented by GenResource in the model). Resources, events and jobs can have parameters (ResPar, DuplPar and JobPar in the model). All parameters have an id and a set of values (setofvals).

The partial order the kernel forms is based on basic assumptions about the model and the connections of the application. A partial order is a binary relation over a set of objects which is reflexive, antisymmetric and transitive, i.e. for all a, b, and c in P, we have that:

- a ≤ a (reflexivity);
- if a ≤ b and b ≤ a then a = b (antisymmetry);
- if a ≤ b and b ≤ c then a ≤ c (transitivity).

The partial order is created by the kernel to create an order for the events to happen. The rules for creating this order are as follows:

- Each block in the application is split up into a start and end event, where end events can only happen after start events (i.e. a Rotate block would generate two events, Rotate_start and Rotate_end, with a relation between the two saying that Rotate_end can only happen after Rotate_start).
- The type parameter of the conditional connection defines how the events of two blocks connected by the conditional connection are connected (see section 5.1.1).
- The conditional connection (section 5.1.1) and the mapping condition (section 5.2.2) both allow for exceptions on the general order depending on job parameters.

A part of the visualization of a generated unfolding of a partial order can be seen in figure 3.3.2.
3.2 Analysis tool(s)

The analysis tool which is currently connected to the kernel is CPN Tools[16], a tool for editing, simulating and analyzing Colored Petri Nets. In CPN Tools a model of the printer data path is generated and by using the analysis options, the optimal path in the petri net is determined. Analysis tools planned to be connected to the DSE tool chain in the future are Uppaal[17] and SDF3[18].

Uppaal is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types. This will most likely be the next analysis tool to be connected to the kernel. SDF3 is a freely available tool which allows users to analyze the performance of Synchronous Data Flow Graphs. These are widely used to model concurrent streaming applications on parallel hardware.

3.3 Visualization tools

The visualization tools of the chain are used to visualize the results obtained by the analysis performed by the kernel and the analysis tool. Currently there are two visualization tools: ResVis (figure 3.3.1) for showing execution times and resource usage and the Kernel itself (with POVis, figure 3.3.2) for showing an unfolding of the partial order of the data path. An unfolded partial order is a partial order in which all parameters (e.g. pagenumber) are instantiated.

ResVis is a tool developed at Océ. With this tool execution times and resource usages can be visualized. ResVis uses trace files as input. These trace files are generated by CPN Tools. With the visualization of ResVis, developers can determine if the designed data path is up to expectations or if certain parts have to improve.

POVis is integrated in the kernel. POVis generates unfoldings of the partial order that follows from the data given by the scenario editor. With this visualization, developers can locate errors in the data path, especially on the connections and ordering.
**Figure 3.3.1: ResVis**

**Figure 3.3.2: part of an unfolding of a PO by POVis**
4. Requirements

This chapter will discuss the requirements that were defined for the scenario editor.

4.1 Input Requirements

As discussed in chapter 2, the idea behind the scenario editor is the Y-chart approach. This means that users of the scenario editor will make an application, platform and the mapping for their data path. This data path can then be analyzed by analysis tools further in the tool chain.

The high level requirements for the scenario editor are the following:

- Users of the scenario editor can graphically define use case steps in the application as blocks.
- Users can make connections between blocks of the application to define an order between the steps.
- Users of the scenario editor can graphically define hardware parts in the platform as blocks.
- Users can make connections between blocks of the platform to show how they are connected.
- The scenario editor allows users to make mappings between application and hardware blocks.
- The same platform or application block is allowed to be in multiple mappings
- The user needs no knowledge of any analysis tool to be able to use the scenario editor.
- The application, platform and the mapping can each be stored in a file for later use.
- The file that contains an application, platform or mapping can be loaded again later to edit the content.

The following requirements came forth during the project:

- The user can define his own attributes for blocks in the application and the platform.
- The application specification can contain concurrent use cases (a use case with multiple jobs which can interleave).

4.2 Interface requirements

The scenario editor is part of a tool chain. This section describes the requirements for the interfaces of the scenario editor with other parts of the tool chain.

- The scenario editor can export the application, platform and mapping data to the kernel for analysis on request by the user.
- The exported data conforms to the interface agreed upon with the kernel (see section 7.1).
- While exporting the data, no information is added or lost.
- If the user requests the analysis of the specified data path architecture, the kernel is started to perform this operation.
- The scenario editor can show the progress of analysis performed by the analysis tool that the user requested (this is done by a shared resource, see section 7.1).
5. Scenario Editor Architecture

At the start of the project, multiple packages and developing environments have been considered to see if they were fit for building the scenario editor. It was determined that Java Eclipse would be the developing environment of choice because of the following reasons:

- There are many packages and plug-ins for Eclipse
- Eclipse is Open Source, so many information is widely available
- Much documentation for support
- Océ also uses some programs in Eclipse
- Java is OS independent

The following alternatives were considered for the scenario editor but discarded:

- Eclipse C++, but because it has less support than the Java variant, it was not chosen.
- Code everything and use graphical packages as OpenGL or DirectX for the interface in C++. This option was discarded because Eclipse and the Eclipse Graphical Modeling Framework have a lower learning curve to get results. On hindsight this was only partially true, see Chapter 9.

When Eclipse was chosen, it had to be determined if, and if so which, plug-in to use. The choice for a graphical editor limited the options to three: the Eclipse Modeling Framework (EMF[12]), Graphical Editing Framework (GEF[13]) and the Graphical Modeling Framework (GMF[14]). The following table shows the differences between the three options:

<table>
<thead>
<tr>
<th></th>
<th>EMF</th>
<th>GEF</th>
<th>GMF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Building tools from structured data model</td>
<td>Building rich graphical editors</td>
<td>Building graphical editors from data models in a structured way</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>XML</td>
<td>No model</td>
<td>Uml-model</td>
</tr>
<tr>
<td><strong>Way of working</strong></td>
<td>Text based programming</td>
<td>Text based programming</td>
<td>Graphical interface</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>Complex</td>
<td>Complex</td>
<td>Fixed but wide options</td>
</tr>
<tr>
<td><strong>Learning curve</strong></td>
<td>Hard to learn</td>
<td>Hard to learn</td>
<td>Easy start</td>
</tr>
</tbody>
</table>

EMF is a collection of packages to create tools from a structured data model. This can be any tool. To make a graphical editor from EMF would mean that extra graphical packages need to be added.

GEF is a collection of packages to create graphical editors. These editors do not need to be structured in any way. To make the scenario editor from GEF would mean that a structure would need to be added.

GMF is a combination of EMF and GEF. It has the structured data model base from EMF and can create graphical editors like in GEF. The only drawback is that GMF is a package of editors so coupled that internal data cannot be edited. The only way to read the internal data is by reading the xml-structured files that can be saved.

Because the goal of GMF is closest to ideas for the scenario editor (graphical and design by building a model) and it was easy to learn, it was decided to use it to make the editor.
The scenario editor is created by combining two separately developed programs. The first program addresses the definition of the application and platform, the second program handles the mapping. The way application, platform and mapping are handled, conforms to the Y-chart approach. For the end user the scenario editor looks like one program.

![Figure 5.1: SE Architecture and how the Y-chart approach fits in](image)

The scenario editor is built up from two programs because of the following reasons:

- The Graphical Modeling Framework (GMF), in which the application and platform program is built, does not allow internal data to be used. To be able to have a program that was capable of using application or platform data, a separate program had to be developed (i.e. the mapping program reads the xml-structured files created by the other program).
- By having two separate programs, it is easier to swap out a part if it becomes outdated.

In this chapter the two programs will be discussed. For both there is a section about the chosen development environment and which primitives the programs contain. Finally there is a section about the connection between the two.

### 5.1 Application and hardware program

The application and hardware program is the part of the scenario editor that will allow the user to define the application and platform. The user-defined application and platform are stored as files (see chapter 8) that can be loaded later for editing or that can be read by the mapping program to create a mapping from.
5.1.1 Application design primitives

The scenario editor allows users to develop printer data paths. Therefore the primitives (core objects) that the developers used to express a data path should be in the scenario editor. These primitives became requirements for the scenario editor. Research learned that the primitives that were needed for the application are two specific blocks (Scanner and Printer), one generic block and two technical blocks (Application Start and Job Information) (also see figure 5.1.1.1 for the corresponding uml model)

Note: all blocks have the attributes Name and Description. They are not in the table to save space.

<table>
<thead>
<tr>
<th>Block name</th>
<th>Attributes</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanner</td>
<td>X Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>Y Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>Nr of channels</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>Channel width</td>
<td>bit</td>
</tr>
<tr>
<td></td>
<td>X resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Y resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Time/line</td>
<td>µs</td>
</tr>
<tr>
<td></td>
<td>Page overhead</td>
<td>µs</td>
</tr>
<tr>
<td></td>
<td>in_Datay size</td>
<td>kb</td>
</tr>
<tr>
<td></td>
<td>out_Datay size</td>
<td>kb</td>
</tr>
<tr>
<td>Printer</td>
<td>X Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>Y Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>X resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Y resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Time/line</td>
<td>µs</td>
</tr>
<tr>
<td></td>
<td>Page overhead</td>
<td>µs</td>
</tr>
<tr>
<td></td>
<td>in_Datay size</td>
<td>kb</td>
</tr>
<tr>
<td></td>
<td>out_Datay size</td>
<td>kb</td>
</tr>
<tr>
<td>General Application Block</td>
<td>Nr of instructions</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>Nr of Channels</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>Channel width</td>
<td>bit</td>
</tr>
<tr>
<td></td>
<td>X Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>Y Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>X Resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Y Resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Pixel size</td>
<td>bit</td>
</tr>
<tr>
<td></td>
<td>in_Datay size</td>
<td>kb</td>
</tr>
<tr>
<td></td>
<td>out_Datay size</td>
<td>kb</td>
</tr>
<tr>
<td>Application Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Job Information</td>
<td>Nr of copies</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>Nr of pages</td>
<td>#</td>
</tr>
</tbody>
</table>

Note: All attributes except Name are optional, this means that not needed attributes can be left empty and they are ignored. Possible problems that may arise from ignored parameters are left to the analysis tools. In the future checks may be added to the scenario editor to avoid possible problems.
The decision to make a *General Application Block* instead of several specific blocks is made because of the following reasons:

- The difference between specific blocks was only one or two parameters.
- Inheritance could not be used between the blocks, as the number of parameters all blocks used in common was small. It would need multiple inheritance which GMF does not allow.
- Specific blocks led to confusion, as different specific operation blocks had the same set of parameters.
- Having a general block increased flexibility for the future.

The General Application Block has all parameters that the specific blocks had. Because all parameters are optional except for *Name*, this does not make it harder for developers to specify the desired operation.

The *Printer* and *Scanner* blocks were added because the technical information (which is platform information) is usually already known at application level. In the old design method for data paths this information was also in the application. For the same reason these blocks have the same parameters in application as in the platform. Later on (in the mapping) there should be checked for consistency between the application and the platform specifications.

![Eclipse UML-model of the application](image)

**Figure 5.1.1.1: Eclipse uml-model of the application**

The technical blocks (*Application Start* and *Job Information*) are created to help the global specification. The *Application Start* block is the start block for the use case specification. From this block the order can be determined by following the connections. In the *Job Information* block information about the job is specified. Originally the *Job Information* block and the *Application Start* block were merged, but when the need for multiple parallel use cases (also called concurrent use cases)
within one application came forth, they were split. One example of a concurrent use case is the use case where a printer gets a print job concurrent with copying another document of different size.

The application has two kinds of connections: the block connection and the conditional connection. Block connections are used between blocks to describe the usual order of the blocks. The conditional connection is there to describe exceptions on the standard order. On hindsight it would have been better to have one connection with the attributes combined, as every conditional connection is serial at the moment. The situation for needing a conditional connection that is not serial is very rare though.

Default attributes: Name, Description

<table>
<thead>
<tr>
<th>Connection name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block connection</td>
<td>Type</td>
</tr>
<tr>
<td>Conditional connection</td>
<td>Condition_name</td>
</tr>
<tr>
<td></td>
<td>Condition_formula</td>
</tr>
</tbody>
</table>

The type attribute of the block connection can be either of the following three: serial, parallel_start or parallel_end. These are for helping the kernel in creating the partial order (PO, section 3.1), see figure 5.1.1.2.

- Serial means that, given the connection A -> B, B can start when A has ended.
- Parallel_start means that, given the connection A -> B, B can start right after A started and end after A ended.
- Parallel_end means that, given the connection A -> B, B can start after A started and B should end before A has ended.

![Figure 5.1.1.2: How the three values of the block connection attribute 'type' (serial, parallel_start and parallel_end) lead to the PO](image-url)
5.1.2 Platform design primitives

With the following six blocks the developers will be able to specify a setup of digital hardware (the corresponding uml-model can be found in figure 5.1.2.1):

Note: all blocks have the attributes Name and Description. They are not in the table to save space.

<table>
<thead>
<tr>
<th>Block name</th>
<th>Attributes</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanner</td>
<td>X Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>Y Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>Nr of channels</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>Channel width</td>
<td>bit</td>
</tr>
<tr>
<td></td>
<td>X resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Y resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Time/line</td>
<td>µs</td>
</tr>
<tr>
<td></td>
<td>Page overhead</td>
<td>µs</td>
</tr>
<tr>
<td>Printer</td>
<td>X Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>Y Page format</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>X resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Y resolution</td>
<td>dpi</td>
</tr>
<tr>
<td></td>
<td>Time/line</td>
<td>µs</td>
</tr>
<tr>
<td></td>
<td>Page overhead</td>
<td>µs</td>
</tr>
<tr>
<td>Bridge</td>
<td>Data Throughput</td>
<td>Mb/s</td>
</tr>
<tr>
<td></td>
<td>Instruction Throughput</td>
<td>Instructions/s</td>
</tr>
<tr>
<td></td>
<td>Overhead/line</td>
<td>µs</td>
</tr>
<tr>
<td></td>
<td>Cores</td>
<td>#</td>
</tr>
<tr>
<td>Operation Block</td>
<td>Speed one direction</td>
<td>Mb/s</td>
</tr>
<tr>
<td>(for hardware that performs operations, i.e. a CPU)</td>
<td>Speed two directions</td>
<td>Mb/s</td>
</tr>
<tr>
<td>Data Transport Block</td>
<td>Max storage</td>
<td>Mb</td>
</tr>
<tr>
<td>(i.e. a bus)</td>
<td>Read speed</td>
<td>Mb/s</td>
</tr>
<tr>
<td></td>
<td>Write speed</td>
<td>Mb/s</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>RAM/ROM/…</td>
</tr>
<tr>
<td>Memory</td>
<td>Max storage</td>
<td>Mb</td>
</tr>
<tr>
<td>(i.e. a hard disk)</td>
<td>Read speed</td>
<td>Mb/s</td>
</tr>
<tr>
<td></td>
<td>Write speed</td>
<td>Mb/s</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>RAM/ROM/…</td>
</tr>
</tbody>
</table>

With regards to the platform blocks, the following points should be noted:

- The Scanner and Printer blocks are specific because of the same reasons as at the application; these are generally known in advance by the developers. The other blocks are abstracted on basic functionality.
  - For abstraction purposes the Scanner and Printer blocks could have been called Data Source and Data Sink, to make the platform less printer specific.
- Because all attributes (except name) are optional, leaving the 'Instruction Throughput' empty for Operation Blocks means that it is an Operation which only uses 'Data Throughput', the same holds the other way around.
  - This can e.g. also be done with a Data Transport Block, leaving the ‘Speed two directions’ open, means the Block can not transport two ways at the same time.
- For the platform there is only one kind of connection, which has as optional parameters 'Name' and 'Description.'
Blocks in the application and the platform can contain custom attributes. This allows the user to expand blocks with special attributes. It gives extra flexibility this way. A custom attribute has an attribute name, attribute type and attribute value. The value is not type checked in the current version.

It might have been nice to define all attributes in the same way custom attributes are defined and have the ‘fixed’ attributes predefined. GMF does not allow this though.

### 5.2 Mapping program

The mapping program lets developers map the application on the platform of the printer data path. This program can load the generated files from the application and platform program part and creates mappings by combining application and platform blocks. During the creation of a mapping, extra information (provided by the user) is
added, which is needed for the analysis tool(s). The mapping describes which tasks of the application will run on which hardware components of the platform.

5.2.1 Development environment
The mapping program was first designed in Java with a simple Java Swing architecture. This allowed for full control of the input, output and looks, but because Swing objects are not allowed in the Eclipse plug-in style that is generated by GMF for the application and platform program, the program design was changed into a Java Eclipse plug-in style[5]. This style is based on views, which show a certain piece of information. This information can be presented in a list, a table or in a tree. More information about the Eclipse plug-in style can be found in the implementation section of the mapping program (section 6.2).

5.2.2 Design primitives
There are less design primitives for the mapping than there are design primitives for the application or platform. This is because most of the information is already in the application or platform. The information that has to be added is about the way the application is mapped on the platform.
For mappings the following points should be taken into account:

- A mapping is one application block on one platform block.
- The order of the mappings in the mapping view is important. The analysis tool will perform the mappings from top to bottom. This can be important for the order of claiming or releasing resources.
- An application or platform block can be related to multiple mappings.
- Every application block (except the Technical blocks, see 5.1.1) needs to be mapped to at least one platform block.

Each mapping has the following attributes:

- The event attribute, which can be either start or end. This is for helping the kernel in placing the claim or release in the right event that is generated from the application blocks (each block becomes a start and end event in the partial order, see section 3.1)
- The claim/release attribute, which can be either of those. This attribute is for setting if this is a claim or a release of the mapped platform block.
- The amount attribute, which is any natural number. This is the amount of the resource the platform block represents that is claimed or released.
- The mapcondition_name attribute, which is the name of the mapping condition.
- The mapcondition_formula attribute, which is the formula of the mapping condition.

The idea behind these parameters is as follows: In an event of a task resources can be claimed or released. In most cases resources will be claimed at the start and released at the end. But to allow for exceptions, the two parameters event and claim/release are kept separate. In case the user wants to claim or release multiple resources in the same event, the user can create multiple mappings. The mapcondition is not completely implemented into the kernel at the moment and is not used regardless of the value, because the right format for conditions is not decided yet. The mapcondition could also hold extra information, e.g. the core of a CPU on which to map the application block.
5.3 Connecting the two parts

The application and platform program together with the mapping program form the scenario editor. The mapping program adds a menu to the application and hardware program to allow the user to open or close the mapping program. The mapping program itself can read the files the user creates (see chapter 8) in the application and platform program and in this way the two parts are also connected.
6. Implementation

This chapter discusses the implementation details of the scenario editor. First the implementation details of the application and hardware program and the mapping program are discussed, then the details about the connection between the two.

6.1 Application and hardware program

This program is built with the Eclipse Graphical Modeling Framework (GMF). This framework allows for easy building of graphical editors in which objects can be handled that conform to a model. This model is an uml-model.

This uml-model (see figure 6.1.1) is designed to be as general as possible, while maintaining the necessary expression power for developers to make printer data paths. The primitives (see section 5.1.1 and 5.1.2) could be placed in the implementation uml-model one-on-one, which makes it easy to understand.

There are a few noticeable differences with the uml-models from section 5.1.1 and 5.1.2:

- The two uml-models are merged into one model, this is to allow for inheritance for the CustomAttribute which is in both application and platform.
- The implementation uml-model (figure 6.1.1) uses less inheritance than the two original models, this is to make the implementation less complex.
- The Scanner and Printer blocks of the platform have the 'hw'-'prefix in the implementation uml-model, as the classes in an uml-model need to have unique names.

The application can use all objects connected to the application class, the platform can use all objects connected to the hardware class. Each of them has as main object a "block" which has several other objects inheriting from this one. They both also have at least one connection. These connections allow the user to connect the blocks to give them an order. In the implementation the connection is represented by an arrow between the blocks.

The output of the application and hardware program are files with an XML structure. This structure is generated automatically by GMF and is based on the uml-model. The structure is described in section 8.2.

More information and detail about GMF, the uml-model and the implementation can be found in Appendix A.
Figure 6.1.1: Implemented Eclipse uml-model

Note: The blockconnectiontypes class (bottom left) is not connected to any other class as this is an enumeration. The attributes of this class are the possible values of the type-attribute of the blockConnection class.
Figure 10.1: Example application in the scenario editor

Figure 10.2: Example platform in the scenario editor
6.2 Mapping program

The mapping program has been designed in conformity to the Eclipse plug-in standard.

Eclipse has Perspectives that define a set of Views that are shown. A View is a window showing a type of information. In Eclipse a View has the format of a Table, a List or a Tree[7].

One example of a perspective is the debug perspective that shows the following Views: Code View, Thread View, Parameter Inspection View and the Console View. Another Perspective can use a different set of Views and Views may be shared over Perspectives.

This is all a user interacts with, but Eclipse developers have to take the following into account:

Internally a View needs a Label Provider, a Content Provider and an Object that serves as input for the providers.

- **Label Providers** transform the basic objects gained from the Content Provider into the information that is displayed by the View. The Label Provider determines in e.g. a Table View which information is in which column.

- **Content Providers** are the providers of the objects the View will show to the user. The Content Provider transforms the (possibly complex) objects in the input to basic objects (usually strings of text or a generalized object), that the Label Provider can handle.

- **Objects** can be any kind of class that inherits from the Object class. Even a custom made class.

![Figure 6.2.1: How concepts of an Eclipse plug-in are connected](image)

For an Object to be presented in a View, it has to be passed through all the parts seen in figure 6.2.1. For the View to update an Object, the View can call a procedure in the List Object, which will call the other parts to do the necessary updates.

In the mapping program the parts are all present and custom made.
The objects that represent the application, platform and mapping information are the ListObjects (inputListObject for application and platform, mappingListObject for mapping). The ListObjects contain procedures to read and write files and to activate the Content Provider. They contain an ArrayList with the objects the List contains. These objects contain the information and have various get and set methods for the fixed parameters of the object.

The mapping program has three views, one for each input type, namely application, platform and mapping. Each has its own Label Provider, but the Content Provider, ListObjects and Objects are the same for the application and platform views. There are a special Content Provider, ListObject and Object for the mapping view, as the mapping information is quite different from the application and platform information.

Eclipse makes a distinction between two kinds of menus: the top menu and the view menu. The top menu is the menu that defines global actions. These global actions are about the perspective’s views as a whole, while the view menu defines action which are about the view itself and the information that the view shows.

The mapping view is the central view of the mapping program. This view has therefore as only view a menu in which actions can be chosen. The view menu actions had to be at view level and not in the top menu, because local information from the view is necessary for the actions.

The mapping view passes information on to the application or platform views if the need arises and the mapping view also gains information about user interactions that are performed on the application or platform view. This is done by making the mapping view a listener of the application and platform views.
6.3 Connecting the two parts

The application and platform program is connected to the mapping program by adding the mapping program as a plug-in to the application and hardware program. Eclipse then automatically adds all actions defined for the top menu bar in the mapping program to the interface of the application and hardware program. To be able to handle the files created by the application and platform program, the ListObjects of the mapping program contain procedures to be able to read these files.

This leads to the user seeing one program that can handle the application, platform and mapping, hiding the two separate programs.
7. Interfaces
This chapter discusses the interfaces that are implemented and considered. First the file interface within the scenario editor is discussed, then the interface between the scenario editor and the kernel, and then the interface between the kernel and CPN Tools. Finally there is a section about threading within the scenario editor.

7.1 Within the Scenario Editor
The interface within the scenario editor is the file interface (see sections 5.3 and 6.3) from the application and platform program to the mapping program. The mapping program has functions to read the files generated by the application and platform program. There is no communication back from the mapping program. More information about the files can be found in the next chapter.

7.2 Between Scenario Editor and Kernel
For the interface between the scenario editor and the kernel two possible interfaces were considered originally: function call and message protocol.

Considered option: message protocol
The message protocol will assume the scenario editor and the kernel are separate programs and uses a socket created by the kernel to send messages between the two. This message system uses acknowledgments after each message and time-outs to avoid errors. There would be a number of messages and acknowledgements that could be sent between the two programs. The message sequence chart for this protocol would look like the chart in figure 7.1.1, except there would be acknowledgements back after each message.

Implemented interface: function call
The implemented interface with the kernel is a function call of the kernel from the scenario editor (see figure 7.1.1). This interface assumes that the scenario editor and the kernel are integrated into one program, with the scenario editor as the controller. This is conform to the requirements set up at the start of the kernel development.

The kernel provides a Java function for the scenario editor to start the analysis: startAnalysis(String xmlPath, String toolName, *JLabel msgLabel);

In this function, there are three parameters to be specified:
- **xmlPath**: the local path of the xml-structured file that the scenario editor generates after user’s editing. In this file (*.se4map, see chapter 8) all the information about the data path the user designed is stored. The kernel can read this file to retrieve the information.
- **toolName**: the name of the tool that the user wants to use for an analysis.
- **msgLabel**: the textArea that shows the messages from kernel to the scenario editor during the analysis. The kernel writes the message directly back to the msgLabel in the scenario editor. The msgLabel is a shared resource and can be updated in the scenario editor while performing the analysis.

*Optional parameter if there is no message back from kernel to scenario editor according to the initial requirement.
For example, if a user wants to use CPN Tools to analyze one existing configuration saved in the xml structured file ‘MyXMLFile.se4map’ (see Chapter 8) and the TextArea to show the message is called ‘messageLabel’, the scenario editor will call the function like this:

```
startAnalysis("C:/DSET/SE xml/MyXMLFile.se4map", "CPNTools", "messageLabel")
```

The complete interface procedure for the scenario editor user is as follows:

1. The user chooses the desired export option (Kernel or CPNTools) in the menu of the scenario editor.
2. A save export file (*.se4map) dialog box appear in which the user can select where to store the export file and to give it a name.
3. When the user has confirmed, the scenario editor saves all application, platform and mapping information in the export file.
4. The scenario editor calls the kernel with the above mentioned function call, with the path to the saved export file as ‘xmlPath’ parameter and the chosen tool by the user as ‘toolName’. The progress dialog box is opened.

After point four the scenario editor is in the same state as before the interface procedure started, progress of the analysis can be tracked in the progress dialog box.
Differences between options
Both options have their pros and cons, which are described in the following table.

<table>
<thead>
<tr>
<th></th>
<th>Function call</th>
<th>Message protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of implementation</td>
<td>Very easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Very easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Error prone</td>
<td>Not</td>
<td>Limited</td>
</tr>
<tr>
<td>Allows for future expansion</td>
<td>Yes, but hard</td>
<td>Yes</td>
</tr>
<tr>
<td>Lockdown of the UI</td>
<td>Can be avoided by putting the Kernel in a separate threat</td>
<td>None</td>
</tr>
<tr>
<td>Coupling</td>
<td>Tight</td>
<td>Loose</td>
</tr>
<tr>
<td>Kernel on other PC</td>
<td>Not possible</td>
<td>Possible</td>
</tr>
</tbody>
</table>

The main reason the function call is implemented, is because this solution gives more benefits (see table) as long as one does not want to keep the kernel and scenario editor completely separate. Also this implementation is far less error prone and could be implemented quicker.

7.2 Between Kernel and CPN Tools
The only interface currently implemented between the kernel and CPN Tools is starting CPN Tools, submitting values via an init file[6] and communicating progress back to the kernel via a socket. The kernel then passes the progress information to the scenario editor via the shared msgLabel resource (see 7.1). There have been plans for other implementations and the most promising implementation[6] is a message sequencing protocol that allows the kernel to communicate with CPN Tools on task level. This means that the kernel would call CPN Tools to perform a certain task and CPN Tools would return a list of possible follow-up tasks after performing the called step. This interface has not yet been implemented.
The file interface that is implemented currently only supports a specific scenario, but will be non-specific in the future.

7.3 Threading
The scenario editor uses a number of threads to perform various parts of the scenario editor. Most of these are automatically generated and handled by Eclipse and therefore will not be described, as these threads can be seen as one thread for both the user and for future maintenance.
The scenario editor itself will start up threads too: it starts a thread for each mapping that the user wants to analyze (the export option) and these threads can overlap. The threads contain the kernel and will start (as separate programs) the analysis tool and ResVis. The threads that are started by the scenario editor are terminated by themselves after they are done with their job. If an exception is raised within the thread, it will close and show an interruption message in the progress dialog box of the scenario editor.
8. Scenario editor files

This chapter will discuss the files generated and used by the scenario editor. First the various file extensions are explained, then the file structure.

8.1 File extensions

All files of the scenario editor are XML based and the four extensions that the scenario editor uses are:

- *.se4_diagram  application information file
- *.se4hw_diagram platform information file
- *.se4map export to kernel file
- *.se4mapping mapping information file

(These files can have any name allowed by the operating system)

All these files except the *.se4map file are used only by the scenario editor itself. The structure of the application and platform information files is linked to the uml model structure (figure 5.1.1) and understanding the model helps in reading the XML structure (see section 8.2).

The *.se4mapping file contains the combined information of the application and platform information files, and the tags for the mapping information.

The file with all information for the kernel (*.se4map) currently has the same content as the *.se4mapping file, but is given a different extension, because this file should contain a correct mapping and to allow for differences between the two in the future (like internal graphical information).

Which parts of the scenario editor use which files and how they are connected can also be seen in figure 8.1.1. Arrows towards a circle mean the program part the arrow points from can write the file, from a circle means it can be read by the program part it points towards.

![Diagram](image)

*Figure 8.1.1: scenario editor (SE) parts using files*
8.2 File structure

Each file always starts with the general XML information which every XML file contains and this will not be displayed in this document. The XML structure is directly taken from the uml model that describes the application and platform (see figure 6.1.1).

The XML structure is created in the following way:

- The classes that contain other classes in the model (application, hardware, applicationBlocks, HWBlock, (hw)connections and customAttributes) are the nodes.
- The classes that inherit from these parent classes are represented by the xmi:type attribute within a parent class node.
- All attributes of the classes and their parent class (in case of inheritance) can be the attributes of the node.
- Each node gets the extra attribute xmi:id with a unique value at generation to keep nodes unique.
- Except for the xmi:type, xmi:id and name attribute, all attributes are optional in the applicationBlocks and customAttribute nodes.
- For connections nodes, all attributes are optional except xmi:type, xmi:id, from and to.

The scenario editor class of the model is not found in the files. The files start at application or hardware level to keep them separate, else they had to be in the same file.

Fragments of possible files can be seen below. It shows at least one example of each tag.

*se4_diagram*

```xml
<se4:application
  xmi:id="_ZEcYwS_pEd6gDqEMyKnVpw">
  <applicationBlocks
    xmi:type="se4:applicationStart"
    xmi:id="_axBPUC_pEd6gDqEMyKnVpw"
    name="Start"/>

  <applicationBlocks
    xmi:type="se4:jobBlock"
    xmi:id="_axBPUC_pEd6gDqEMyHfJpo"
    name="Job1"
    Nr_of_Copies="1"
    Nr_of_Pages="9">
    <hasCustomAttribute
      xmi:id="_axBPUC_pEd6gDqEMyPvIxu"
      Attribute_Name="page_format"
      Attribute_Value="A4"
      Attribute_Type="String"/>
  </applicationBlocks>

  <connections
    xmi:type="se4:blockconnection"
    xmi:id="_dzDc8C_pEd6gDqEMyKnVpw"
    from="_axBPUC_pEd6gDqEMyKnVpw"
    to="_axBPUC_pEd6gDqEMyHfJpo"
    type="serial"/>
</se4:application>
```
The file structure of the platform file is basically set up in the same way as the application file. For comparison purposes, the hwblocks nodes are comparable to the applicationBlocks nodes and the hasHWConnections nodes are comparable to the connections nodes of the application.

*.se4hw_diagram
<se4:hardware
 xmi:id="_n28DkTroEd6s-du3QBL2mQ">
  <hwblocks
   xmi:type="se4:Operation_Block"
   xmi:id="_p7LMMDroEd6s-du3QBL2mQ"
   name="CPU" Cores="3"/>
  <hwblocks
   xmi:type="se4:Data_Transport_Block"
   xmi:id="_sNWDIDroEd6s-du3QBL2mQ"
   name="CPU-bus"/>
  <hasHWConnections
   xmi:type="se4:hwConnection"
   xmi:id="_yKy5ADroEd6s-du3QBL2mQ"
   from="_p7LMMDroEd6s-du3QBL2mQ"
   to="_sNWDIDroEd6s-du3QBL2mQ"/>
</se4:hardware>

The files that contain mapping information are set up in the following way:

*.se4mapping & *.se4map
These files have a general structure:
<se4:application>
 Applicationtags
</se4:application>
<se4:hardware>
 Hardwaretags
</se4:hardware>
<se4:mapping>
 Mappinginfo
</se4:mapping>

The applicationtags and hardwaretags are as in the two sections (se4_diagram and se4hw_diagram) above. The mappinginfo contains tags which are as follows:
<mappingBlock
 xmi:type="se4:mapping"
 xmi:id="_map0001"
 fromApp="someAppID"
 toHW="someHWID"
 event="start"
 claim="true"
 amount="10"
 mapcondition_name="only first copy"
 mapcondition_formula="C=1"/>

A mappingblock tag has a xmi:type and id, followed by the ids of the objects mapped. All other parameters are the parameters the user fills in while creating the mapping (see section 5.2.2). For a mapping block all parameters are mandatory.
Note that unlike in the se4_diagram and se4hw_diagram files, the `<se4:application>` and `<se4:hardware>` have no id, just as the `<se4:mapping>`.

### 8.3 Options file

To allow the scenario editor to know where the user installed CPN Tools and where the user placed the CPN model, the scenario editor will generate an ini-file (SEsettings.ini) on first start up of the program with the default paths (hard coded) to CPN Tools and the CPN model. This is necessary as the kernel will also read this file for this information. When this file is present at start up, the scenario editor will read the last saved path from this file. The paths can be changed by the user in the scenario editor in the mapping view under options.
9. Evaluation

This chapter will evaluate the results from the graduation project. First the scenario editor is discussed, then Eclipse and finally the maintainability of the scenario editor.

9.1 Scenario editor

The tool chain is created for the design space exploration of data paths. Developers can create data paths in a graphical way with drag and drop blocks. While the chain as a whole certainly has not reached its full potential yet (mostly because of the weak interface between the kernel and CPN Tools, see section 7.2), its future power can already be seen. When more analysis tools are connected and the connection between the scenario editor and the kernel is strengthened with the conditions of the conditional connection of the application and the mapping condition of the mapping (these are not used by the kernel currently), it will allow developers to perform multiple analyses in rapid succession. The analysis power of the chain becomes even stronger when the chain gets the option to make parameters deviate, so the analysis tools can find the best value (also see section 10.3.5).

The scenario editor itself already is stable, there are no crashes and it fulfills the requirements set at the start of the project. A real printer data path use case and platform was taken and made in the scenario editor without problems as long as the current limitation of static use cases (no scheduling) only was kept in mind. This is a severe limitation in itself, but this limitation will be lifted in the future. The mapping part could use some improvements on the graphics, but all the required functionality is in place and it can be expanded in the future.

9.2 Eclipse

The Eclipse GMF is a powerful tool for quickly generating graphical editors. A developer that understands GMF can make a new (application and platform) editor depending on the special graphical wishes (such as colors and constraints of objects) within two days.

The mayor drawback of GMF is, that when something in the model has to change (as a new block or new attributes), and one has to do almost everything again from scratch. Another flaw is that the automatic generation of the mapping model (see figure 2.1, Appendix A) has bugs and has to be done by hand. This means the developer has to check all generated parts and correct mistakes, which is extra work and therefore increases the time to create a new editor.

Big plusses of GMF are the customizable looks of the program, the many automatic generation functions and the broad support. GMF allows developers to very quickly make a graphical editor, not many other development environments have this feature, while remaining flexible.

For developing plug-ins and general Java programs, Eclipse is a very nice environment. It has easy accessible auto-complete functions, handles many imports on his own and has a very good debugger. Building programs in Eclipse is a nice experience.

Both GMF programs and Java plug-ins can be made standalone in a Rich Client Platform[8]. This can be done very easily, but RCP has one mayor drawback: it does not support objects of the Java Swing graphical package. Objects of the Java AWT graphical package can be used however. Java Swing objects are easier to use, while AWT objects require more configuration to set up.
9.3 Maintainability

The application and hardware program has a maintainability which is not good when new blocks or attributes have to be introduced. If they have to be introduced, then the complete editor has to be built again from almost scratch, as the model has changed. This means all files that are depending on the model (see figure 2.1 in Appendix A) have to be changed too. This process does not take long if there are few customizations (about half a day for an experienced GMF developer), but becomes a harder and longer process if more special features are introduced, like block colors and constraints. Explanations about how to add blocks, attributes and special features can be found in Appendix A.

Maintainability for changing special features or introducing new special features (e.g. the color of a block) is very good. These can be changed quickly and other parts will not be affected, only the code generator has to be generated again, which takes five minutes.

The mapping program consists of Java code and is therefore easily maintainable for experienced Java programmers.
10. Conclusions
This chapter will discuss the conclusions of the project. First the the goals are discussed, then the impact and finally some future work is discussed.

10.1 Goals
The goal of the scenario editor is to support developers in creating their printer data paths. The scenario editor does allow this in an easy and understandable way. All requirements (see chapter 4) are fulfilled by the current scenario editor. Making specifications for the application and platform has been tested by taking a real printer data path use case and platform and creating them in the scenario editor. This could be done without problems as long as the current limitation of static use cases (no scheduling) only was kept in mind.

10.2 Impact
The current version of the scenario editor and especially the tool chain as a whole will need extra development on the interface to the analysis tool(s) before putting into practical use, as there is no general interface between the kernel and CPN Tools (see section 7.2) at the moment. The scenario editor itself can be used already by developers to design simple static printer use cases and to get an idea of the direction that future development has to take. Because the chain has no complete connection to any analysis tools at the moment, there is no big impact.

10.3 Future work / Change cases
10.3.1 Using the programs for data paths of other products
At the moment the scenario editor is set up for designing printer data paths. This can be seen in the scanner and printer blocks and in the job information block (which has the attributes number of pages and number of copies). It can also been seen in some of the attribute names (as time per line). It can be imagined that in the future the same approach of using a tool chain with the Y-chart approach will be used for developing data paths of other products such as televisions or DVD players. If the scenario editor is to be rewritten to design other data path types, the only part that has to be changed is the application and hardware program part. This program part will need to contain new blocks and attributes that support the new type of data path. This can be done within a few days. For the mapping program part, only the read and write procedures will have to change to support the (most likely) new file extensions and keyword changes.

10.3.2 Adding scheduling algorithms
Decision making is an important aspect of data path analysis. If multiple blocks can be chosen to perform their action (like scanning and printing), which one will go first? How the order is chosen is decided by the scheduling algorithm. One example of a scheduling algorithm is First Come First Served (FCFS), which says that actions are performed in order of arrival. Scheduling algorithms for scheduling the tasks or events can be placed in both the application and the mapping part of the data path. Which of the parts to extend with scheduling algorithms should be decided in the future. Scheduling algorithms that determine the scheduling of a resource should be placed in the platform.
When either of the parts is extended to allow for scheduling policies, the policies can be either presented by a set of fixed scheduling algorithms (i.e. a dropdown box) or the user can be allowed to make his own scheduling rules. In the latter case, a format for custom scheduling rules has to be determined. This format can either be in the language of the tool to be chosen analysis tool (which is not desirable) or some currently unknown high level language.

The simplest option, when scheduling policies need to be implemented on the application part, is adding them as Custom Attributes. Then the attribute name could be “scheduling algorithm” and the attribute value the scheduling policy (either fixed or custom, see above). This option is already in place and it will be passed on through the kernel. Then in either the kernel or the analysis tool there has to be checked for this attribute.

The other option is adding a new global attribute, which causes the need to update the mapping program and kernel interface for a new XML format. This is also the case if the scheduling policy needs to be added to the mapping part. Independent of which option is chosen; it has to be taken into account that scheduling algorithms might raise the need to also add priority numbers to blocks or connections. If these have to be added to connections, the tools will have to be extended to add the extra parameter. On blocks this could be resolved via the Custom Attributes.

If there is chosen for the fixed set, the kernel or analysis tool has to be able to convert the given value to the required scheduling policy. When there is chosen for new scheduling rules, the scenario editor, kernel or analysis tool needs to be able to interpret the scheduling rules.

### 10.3.3 Showing connections

In the future the option to show the application connections in the mapping program part might be considered. In the mapping program code of the old Java Swing version (see section 5.2.1), methods can be found that allowed to show the connections in the application view. In the new version, a few things would have to be changed to get it to work:

- The code that detected connections did not allow for loops in the application specification. With the new conditional connection, loops can exist. This can easily be solved by not adding the next blocks to the list of “to consider blocks” if the connection is a conditional one.
- Now both the blocks and the connections have to be passed through to the Content Provider and the Label Provider, so their code will also have to change.

### 10.3.4 Feedback

As could be seen in figure 3.1, the tool chain was designed with the idea for feedback from the analysis or visualization tool to the scenario editor. This is currently not supported, but there are a few possible ways to do this:

- Monitoring progress messages.

  The analysis tool could send back feedback via the progress messages which are implemented. The scenario editor would have to monitor the progress messages and look for a certain tag that would be chosen to differentiate the progress messages from the feedback messages.

- Adding a feedback channel.

  In the same way the progress message channel is added in the interface to the kernel (section 7.1), there could also be a feedback channel.
- Feedback files
  The analysis tool or the visualization tool could generate a feedback file in a format to be determined, which is checked by the scenario editor after performing analysis to see if any feedback emerged. Feedback that could be given by the analysis or visualization tool can be e.g. bottlenecks, possible improvements or found optimal values. This feedback can be presented by the scenario editor either via messages (which can be easily be implemented should either of the first two options be chosen) or via graphical emphasis. The latter can only be realized if the user requests for it, as the actions of the user could be interrupted if this would be done automatically after analysis is done and also because the user can close the application, platform or mapping while analysis is performed.

10.3.5 Statistics as parameters
In the future the wish may arise to allow ranges of values to be the input for certain parameters to let the analysis tool find the optimal value or to quickly test multiple options. Possible solutions for allowing ranges of values for parameters can be:
  - Define a new type in GMF that can be used for these parameters.
  - Make a special export function in the scenario editor that allows the user to select parameters and set ranges for them in a special dialog box.
11. Literature & Links


12. Appendix A: Eclipse GMF Maintenance guide

Scenario Editor

Application and hardware program plug-in

Eclipse Graphical Modeling Framework
Maintenance guide

Sander van Zuidam
December 2009

Eindhoven University of Technology
0. Index
1. Introduction .................................................................................................................. 44
2. General GMF build up .................................................................................................. 45
3. Domain model ............................................................................................................. 45
4. Tooling Definition Model .......................................................................................... 47
5. Graphical Definition Model ....................................................................................... 47
   5.1 General structure .................................................................................................. 49
6. Mapping Model .......................................................................................................... 49
7. Diagram Editor Generator Model ............................................................................. 50
8. General Tips ................................................................................................................ 51
1. Introduction

This document is written to give the reader a better understanding of the Eclipse Graphical Modeling Framework (GMF). It can also be used for maintenance of the application and hardware program plug-in of the scenario editor made for the Octopus project.

To use the GMF, a GMF project must be made in Eclipse via File -> New -> GMF project. This creates the basic files that are needed to make a GMF based editor.

If the reader is completely new to GMF, it is strongly advised to read the GMF tutorials on the internet (http://wiki.eclipse.org/index.php/GMF_Tutorial) for a better understanding.
2. General GMF build up

A GMF project consists of a number of files that remind of the Y-chart approach of the scenario editor in general. These files and their relations can be seen in the GMF Dashboard (see figure 2.1). All files can be created by either choosing “Create” on the dashboard or via File -> New -> Other and looking in the Eclipse Modeling Framework and Graphical Modeling Framework sections.

![GMF Dashboard Diagram](image)

**Figure 2.1: the GMF dashboard**

As can be seen on the Dashboard, a GMF project starts with a Domain Model. This model is transformed to a Domain Gen Model, which can be used to make the model and edit code. After this the Graphical Definition Model, which holds all graphical information and the Tooling Definition Model, which holds the menu information, should be made. All files so far created will then be combined into the Mapping Model. From this Mapping Model file a Diagram Editor Generator can be generated, which will generate the diagram code. This diagram code, combined with the earlier generated model and edit code is the actual source code of the program.

When RCP is checked on the dashboard while doing the transformation, the program will run in the Rich Client Platform (RCP). It will then run stand alone instead of running in an eclipse environment.

3. Domain model

A domain model file (*.encore) will eventually contain all objects and the relations between them, that should be in the editor. The encore file is usually already present when a new GMF project is started, but can always be created via the New -> Other... menu option, where it is called an Encore Model.

To help making a domain model, it is advised to “initialize an encore_diagram” from the domain model (from the right click menu of the encore file) and make the model inside this diagram. The encore_diagram is set up like an uml-model. There must be one top element and all other elements will have some relation to the top element. The encore_diagram of the application and hardware program can be seen in figure 3.1.

This is the file that should be changed first should the need arise to add a new block or new attributes to the application or platform.
Figure 3.1: uml-model

Note: The blockconnectiontypes class (bottom left) is not connected to any other class as this is a enumeration. The attributes of this class are the possible values of the type-attribute of the blockConnection class.
The application diagram can use all objects connected to the application class, the platform diagram can use all objects connected to the hardware class. Each of them has as main objects a “block” which has several other objects inheriting from this one. Also they both have a connection. This connection allows the user to connect the blocks to give them an order.

Objects are made using the EClass object. The connections that can be used are Association and Aggregation relations. Also the Generalization option can be used to make objects inherit from others.

To create new attribute types with fixed values (which will be a dropdown box in the implementation), use the EEnum object and select the EEnum name in the attribute type to use the EEnum values. An example of this can be seen in figure 3.1 with the blockconnectiontypes class. It holds three values which are the possible values for the type attribute of the blockconnection class.

A new block is added by adding a new EClass, giving it the name of the new block and letting it inherit from the right “superblock” (ApplicationBlock or HWBlock). New attributes can be added by adding an attribute to the EClass that should have the new attribute. This attribute needs a name and a type.

### 4. Tooling Definition Model

The tooling definition model is straightforward. This file (*.gmftool) lets the developer determine how the menu to the right of the canvas looks and which objects the user can create from there. Separators and labels can be added at any location, using the right mouse button menu. For an example, see Figure 4.1

![Figure 4.1: Example Tooling Definition Model](platform/resource/SE4/model/set.gmftool)

### 5. Graphical Definition Model

The Graphical Definition Model (*.gmfgraph) contains all graphical information that is needed. This includes the forms and colors of the various objects created, the labels that will appear on the objects and the containers. While making this file, it is advised to think in advance of what attributes to be visible on the objects. Attributes that should not be visible, should not have a label.
By default all attributes will get a label, but it is much less work to remove the checks on the generation wizard then removing labels later.

In general the graphical definition will contain the following things:

- Figure Gallery, which holds all Figure details.
- Nodes, which are references to the objects of the Domain Model.
- Connections, which are references to all connection-like objects in the Domain Model.
- Compartments (optional), which are references to objects that be contained inside other objects.
- Diagram Labels, which are references to attributes of objects.

In the Figure Gallery there will be Figure Descriptors with a shape descriptor and Child Accesses to each of their labels. The shape descriptor will contain labels and a layout (see Fig 5.1). It is advised to look at all options of objects that can be added to the various points to get a feeling for the options.

For starters with GMF and the Graphical Domain Model, keep the following things in mind:

- For the connections, it helps adding arrows to the ends of the line, as a connection is just a flat line by default. This is achieved by adding a polyline decoration to the figure gallery (which is an arrowhead by default) and adding this decoration to the beginning or end decoration of the connection in the Mapping Model.
- When planning to make certain figures exist inside other figures, the figure that is inside the other needs a compartment on the canvas.
5.1 General structure

Nodes or Connections (depending on whether it is a connection or not) point to Figure Descriptors in the Figure Gallery. Diagram Labels point to the Child Access items in the Figure Descriptors. These Child Access items point to the Labels in the Shape item (like Rectangle in figure 5.1) in the Figure Descriptor. In these Labels one can set the default text.

![Diagram](platform://resource/SE4/model/se4.gmfgraph)

**Canvas se4**
- Figure Gallery Default
- Node ApplicationBlock (ApplicationBlockFigure)
- Node ApplicationStart (ApplicationStartFigure)
- Node Scanner (ScannerFigure)
- Node Printer (PrinterFigure)
- Node General_Application Block (General_Application_BlockFigure)
- Node CustomAttribute (CustomAttributeFigure)
- Node JobInformation (JobInformationFigure)
- Connection Connection
- Connection ConditionalConnection
- Connection BlockConnection
- Compartment: CustomAttributeCompartment (CustomAttributeFigure)
- Diagram Label ApplicationBlockName
- Diagram Label ApplicationBlockDescription

*Figure 5.1.1: Showing all objects on the Graphical Definition*

6. Mapping Model

The mapping model (*.gmffmap) is the place where everything comes together. Here it really is determined which objects will be generated when something is chosen from the menu. Here also input formats, label looks and restrictions can be determined.

Sadly, the mapping model generator has many bugs, and you will have to check everything generated for errors. Also the generator only adds the first attribute and does not do anything about compartments.

When the New Mapping Model wizard has been completed you will have a mapping that contains Top Node References (for each object you made) that have Node Mappings and a Link Mapping for each connection. Each Node Mapping will have one Feature Label Mapping.

The Top Node Reference is always correct, but the rest usually isn’t. From the Top Node Reference it can usually be seen which object it is about. This object needs to be set in the Top Node parameters ‘Diagram Node’ and ‘Tool’. Also the correct Label needs to be set as Diagram Label in the Feature Label Mapping.
Attributes will only update on the figure if the attribute has a Feature Label Mapping. The attribute will appear with default text with just the label in the Graphical Definition Model.

It is advised to use the Refresh option from time to time to update labels of Top Node References and Connections. When done, use the Validate option to see if any errors exist.

7. Diagram Editor Generator Model

This model (*.gmfgen) is the place where all kinds of technical settings can be changed like the file extensions and version number. Usually there is no need to change many settings here. One setting that should be changed at all times is the “same file for diagram and model” parameter in the Gen Editor Generator. This parameter should always be true, else information will be missing for the later parts of the scenario editor.

When working with two diagrams generated from the same encore model (as application and platform are of the scenario editor), the following parameters should be changed in the second diagram editor generator model:

- “diagram file extension”, “domain file extension”, “model ID” and “package name prefix” in Gen Editor Generator
- “ID” and “name” in Gen Plugin.

If these are not changed, the two diagrams will overwrite parts of their code which will lead to errors.
8. General Tips

For starters GMF can be quite complex without outside help. These are some general tips and comments to help new programmers with GMF:

- None of the files generated or the domain model name is allowed to start with a capital. If they start with a capital, this can lead to strange errors in the code generated by the diagram editor generator model.

- Think in advance about how to make the domain model. When the need to change it arises during the making of later parts, all the work done so far will need to be done over.

- Copy paste parts of the files and then adapt the copied part to the new situation almost never works; there are hidden links that are copied also and not changed. Therefore it is advised to always use the right mouse menu to create the parts instead of copying them from a similar case.