Routing Information and Presentation

Bas Luksenburg

April 2012
Eindhoven University of Technology
Department of Mathematics and Computer Science

Routing Information and Presentation

Bas Lukenburg BSc
Eindhoven, April 2012

Supervisor:
dr. G.H.L. (George) Fletcher (TU/e - W&I - DH)

Tutors:
dr. G.H.L. (George) Fletcher (TU/e - W&I - DH)
Ing. J. (Johannes) Mulder
  (ASML Holding NV - Senior Designer and Integrator)
R. (Rene) Fussenich MSc
  (ASML Holding NV - Industrial Engineering Design for
   Availability System Level Diagnostics Engineer)

Assessment committee:
dr. M. (Michael) Westergaard (TU/e - W&I - AIS)
dr. G.H.L. (George) Fletcher (TU/e - W&I - DH)
dr. T.G.K. (Toon) Calders (TU/e - W&I - DH)
Ing. J. (Johannes) Mulder
  (ASML Holding NV, Senior Designer and Integrator)
R. (Rene) Fussenich MSc
  (ASML Holding NV, Industrial Engineering Design for
   Availability System Level Diagnostics Engineer)
Abstract

ASML is the world’s biggest producer of lithography machines; machines that sometimes break down. Service engineers then have to repair those machines and need access to information about the routing of the cables in the machines to repair them, but this information is scattered over several digital and analogue data sources that are hard to search through. For this master project, research was performed to see if it is possible to develop a software framework that automatically imports the routing information from these sources to construct usable maps of the cables. Furthermore, these sources are prone to errors and some known errors have to be detected and corrected or highlighted by the program or ignored without crashing.

The focus of the project has been put on feasibility research and the raising of enthusiasm. This means that there were two important focus points: Firstly this means investigating all available data sources to find out if and how the data can be automatically imported, what kind of software framework is needed for this and the difficulties this will raise. Secondly a demo tool was needed as proof of concept to raise enthusiasm about this project at ASML and to demonstrate the possibilities. The software framework thus had to be implemented to be able to carry out a large part of the described importing, handle the difficulties and to show the usefulness in an intuitive tool.

In the first phase of this project, a study on deductive databases to use for the imported dataset and following queries was conducted, showing that there are usable databases that use Prolog. However, Oracle in combination with JAVA was preferred as it already belongs to ASML’s set of companywide available software.

The import functionality can import several data sources into the database. This project has proved that it is possible to extract all needed data, but the extraction of some data (especially lengths from EDArch files) still has to be implemented. The sources that are used are: Cable Overviews in Excel format for cables and connectors, files out of the TCE database for lengths of cables, and files out of the EDArch for information on signals, connector pins and the lengths of circuit boards. The tool itself momentarily imports a test set that contains all the needed information in one Excel file, to be able to spend more time on the implementation of the rest of the tool.

To be able to handle the error prone files, a set of checks is implemented that check and correct or highlight the following exceptions:

- Spider cables
- A signal changes its name within a location
- There are multiple pins for one signal on a connector
- Mismatched genders of pairs
• A general breach in the path of a signal by:
  o Discontinuity between CO’s
  o Unforeseen changes in Xnr or 12NC of two connected connectors.

• Cables that have only connector

• Cables that have multiple versions

The functionality that the user will use most is implemented in a standard searching facility in which the user can search for 12NCs, SAP names and signals. It returns properties of cables and connectors and for a signal the map of the signal can be generated. This map consists of a path of segments the signal runs over with the cumulative length and the component’s properties next to them.

The developed application consists of a 3-tier architecture with the layers: Database, Data and application logic, and Presentation. The Model-View-Controller pattern is used to decouple the data and the view. The decomposition in packages is done according to this pattern.

This forms an application that will take care of the whole automation of getting from the data sources to a usable map of a signal that can be used in combination with a TDR tool to find a breach in seconds and reduce repair times drastically.

In the future, the tool will have to be improved to become more mature, so it can be used “in the field” by the service engineers. Some of the most important steps that need to be taken are the full implementation of the importing of all data sources, including lengths from EDArch, and the implementation of a solid form of version control and database management from within the application.
# Table of contents

ABSTRACT .................................................................................................................. 3

TABLE OF CONTENTS ............................................................................................ 5

1 ACKNOWLEDGEMENTS ......................................................................................... 7

2 INTRODUCTION ..................................................................................................... 8
   2.1 ASML .................................................................................................................. 8
   2.2 CONTEXT ........................................................................................................... 8
   2.3 PROBLEM DESCRIPTION ............................................................................... 11
   2.4 OVERVIEW OF THE REPORT ........................................................................ 11

3 PRELIMINARIES ................................................................................................... 12
   3.1 INTRODUCTION ............................................................................................... 12
   3.2 RESEARCH ON DEDUCTIVE DATABASES ....................................................... 12
      3.2.1 Introduction ............................................................................................ 12
      3.2.2 Criteria .................................................................................................. 12
      3.2.3 Search .................................................................................................... 12
      3.2.4 Quick overview of possible databases ................................................. 13
      3.2.5 Database details ..................................................................................... 13
   3.3 CONCLUSION .................................................................................................. 14
   3.4 FOLLOW UP .................................................................................................... 14

4 APPROACH ........................................................................................................... 16
   4.1 INTRODUCTION ............................................................................................... 16
   4.2 REQUIREMENTS SPECIFICATION ................................................................. 16
   4.3 LOOKING BACK .............................................................................................. 16

5 DESIGN APPROACH ............................................................................................. 17
   5.1 INTRODUCTION ............................................................................................... 17
   5.2 STRUCTURE OF THE DESIGN ....................................................................... 17

6 IMPORTING ........................................................................................................... 19
   6.1 INTRODUCTION ............................................................................................... 19
   6.2 CABLE OVERVIEWS ...................................................................................... 19
   6.3 ELECTRONIC DESIGN ARCHIVE (EDArch) .................................................. 21
   6.4 TEAMCENTER ENGINEERING (TCE) ............................................................. 25
   6.5 SAP .................................................................................................................. 27
   6.6 TEST SET ........................................................................................................ 27

7 REPRESENTATION OF THE DATA ........................................................................ 29
   7.1 INTRODUCTION ............................................................................................... 29
   7.2 DATABASE STRUCTURE ............................................................................... 29
   7.3 THE DATA MODEL .......................................................................................... 32

8 DATA INTEGRITY AND EXCEPTIONS .................................................................... 35
1 Acknowledgements
I completed this thesis as part of the Master’s programme Business Information Systems at the Eindhoven University of Technology and am grateful that I could do this by solving a real-life problem at ASML. I had the help of a lot of people while completing this project:

Firstly, I want to thank my supervisors:

George Fletcher for his support and feedback during the project. His friendly character, support, availability, and feedback have all been very helpful and his faith in my abilities helped me bring this project to a good end.

Johannes Mulder and Rene Fussenich for their support at ASML, their enthusiasm with which they mentored the project and their continuous availability to give feedback and constructive criticism, without them the project would have been a lot harder.

Outside this project I would like to thank:

My parents Petra and Hans Luksenburg, who unconditionally supported me throughout my thesis, study, and the whole of my life and I could not have come as far as I am now without them.

My lovely girlfriend Tara who supported me, improved my English, but most of all gave me the energy and joy I needed to be able to fly through these past few months. I love you with all my heart.

And last, but not least: My dear friends, from the university and my hometown, who made the past six and a half years go by like a breeze. They were willing to help at difficult moments and caused much chaos, laughter and overall happiness. Thank you all.

Bas Luksenburg

March, 2012
2 Introduction

2.1 ASML
ASML is the world’s leading provider of lithography systems for the semiconductor industry, manufacturing complex machines that are critical to the production of integrated circuits or chips. Headquartered in Veldhoven, the Netherlands, ASML is traded on Euronext Amsterdam and NASDAQ under the symbol ASML.

2.2 Context
The machines that ASML produces will, as all machines do, sometimes break down. A service engineer then has to go to the customer to repair the machine, while the company loses thousands of dollars due to the temporary production stop. Once it is clear that the fault is in the machine itself, it needs to be repaired.
Reparation can take a couple of hours up to a few days, because the service engineer has to locate the route of the signal related to the error and check all the parts for faults.

Such a machine consists of different modules that are connected to each other by many and lengthy cables. A module for instance can be the set of parts to handle all movement of the wafers. The module itself contains multiple smaller projects that perform a specific function within the module like the wafer handler and the wafer stager for respectively preparing the wafers in the machines and exposing them. Each project can be divided into smaller components like sensors, circuit boards and cables. These components will be checked for errors and replaced when necessary.

All these parts also have connectors to allow them to be connected to each other. Each connector has an X-number to identify the connector on the location it resides on and each connector consists of a number of pins. These pins are numbered and almost always form pairs.

For the sake of simplicity each machine will be considered as just a collection of cables that connect several parts to each other. A list of these interconnections will be called a Cable Overview (CO) and is available on three levels.

The first and highest level is the Top CO, which is an overview of the entire machine. It does not store any interconnections itself, but it stores the collection of lower level cable overviews that all together form the machine. It is comparable to an index in a book to show what is connected to what to easily find all the subsections.
A lower level 3D CO stores the cables that interconnect the modules in the machine and other cables that could not be defined in the lower level Project CO. This Project CO is the lowest level cable overview and defines the cables that exist between the components within a project module itself. The 3D and Project COs are complementary to each other and together form a module.

For this project, we are especially interested in the electric signals that are sent through the machine. Such a signal can be measured with a newly introduced tool (Time Domain Reflectory) that uses an echo to measure the length of the signal’s path. If this length can be checked against the length that the path should have, the service engineer will immediately be able to locate defect, since the echo will return there.

Each signal is easy to distinguish, because it has a unique name, a unique and almost always forms a pair with a return signal that follows the same path, but in the other direction. It is common to give such a pair a name like SIGNAL_NAME_X, where the X is used to differentiate between the two signals, for example: SIGNAL_NAME_P and SIGNAL_NAME_N for a positive and negative signal.
Much like an IDE cable as in figure 2-2, each cable in the machine consists of one or more wires, connected to a pin on each side. The unique path of a signal thus goes through of a set of interconnected pairs of wires. Such a pair of wires with a pair of pins on the connectors on each side of a wire will be called a segment. A cable will thus consist of one or more segments. The pair of pins on a connector will be called a node, so a segment will have two nodes.

Figure 2-3 A segment is a pair of wires with a pin on the start and end of each wire. Each wire belongs to a single signal.

The path that is of interest to the service engineer will consist of one or more segments interconnected by nodes, like this: start node - node - segment - node - ... - node - segment - node - end node. Only the start and end node can be connected to a component that is not considered a cable, like a sensor or motor. All other components that are used in the path of a signal (like a cable or iPCB) that are not on the start or end node will be considered cables.

Figure 2-4 A path formed by a start node, several segments and an end node.
In the final application this kind of path will need to be represented as a map containing the segments and the length of every segment. The delivered software framework will be called RIP (Routing Information and Presentation)

2.3 Problem description
The main problem is, as described in the context, finding the routing (and length) of a signal that is measured with the TDR tool, to find the location of the defect in the machine. This project, called RIP (Routing and Information Presentation), should thus provide an easy way to find the length of the cables on the path. The route that the signal follows should also be visible, so the length can be shown alongside the cables on this route (like a road map). This way the service engineer can easily find out which cable is defective and at what point the cable is probably broken. The information about the components connected to this cable should also be accessible through this application.

The routing information that is needed is already present, but scattered over several sources that are hard to access and search through. These sources are sometimes made by hand, which makes them very prone to errors. Some well-known and identified exceptions have to be detected and corrected, highlighted or explicitly ignored by the program.

The goal of this project is to develop a new database and Application Programming Interface (API) in the form of a JAVA library on top of that to provide the aforementioned service. Also a demo tool will be delivered to provide a proof that the API can be used for this purpose.

2.4 Overview of the report
In chapter 3 the introductory research on finding a suitable deductive database system for this project will be explained. Chapter 4 explains the process of getting the user’s requirements to a system specification and chapter 5 describes the general design approach to the problem. In chapter 6, the importing of files will be explained and chapter 7 will go deeper into the representation of the data in the database and the data model of the application, while chapter 8 does the same for the data integrity. Chapter 9 is about the application logic that handles the user’s actions in the data model and the whole of chapter 10 describes the user interface. In chapter 11, the results are summarised and an indication of future work that is necessary will be given.

The first appendix (A) summarizes the task analysis that was done, while appendices B, C and D summarize respectively the user, system and architecture design documents. In the last appendix (E) the definitions are given for the most used terms in this document.

---

1 http://www.flickr.com/photos/travelinlibrarian/273079025
3 Preliminaries

3.1 Introduction
In this chapter, the research is described that had to be done before starting on the actual solution to the problem. This research compares different kinds of Prolog databases that are made for handling recursive queries and concludes that the IRIS database and after that the DLV database would be the best databases for this project. In the end, however, the choice was made to use the standard technologies used by ASML for maintainability reasons. This solution consists of an Oracle database with JAVA doing the recursive logic. For the future, the study indicates that there are very useful tools for this kind of data, when RIP is scaled up to a larger type of environment. It would be useful to investigate further possibilities when the need for this upscale arises.

3.2 Research on deductive databases

3.2.1 Introduction
The research problem in this master project is of a kind that will require the storage of many edges and path finding queries, which brings the need for recursive queries to mind. Queries like this will answer questions, such as ‘which points are the ancestors of this point’. Though this can be done in a standard query language like SQL, other options that are specifically designed for these problems need to be investigated. The Prolog language, for instance, seems very suitable for this problem as it describes dependencies and can answer these kinds of questions in a natural way. Because we execute the queries on data in a database, it seems to be a good idea to investigate Datalog, which is a subset of Prolog for deductive databases.

3.2.2 Criteria
Based on the environment within ASML and the wishes of the client, the systems will have to satisfy the following criteria:

1. The presence of a JAVA API, because the GUI software and API will be written in JAVA.
2. Support of a MySQL or Oracle database, because these are the commonly used databases within ASML.
3. The presence of a license that permits us to use the software for free within a closed source application.
4. The presence of a stable, mature version.
5. Easy to install, configure and maintain, if it is to be used on a lot of machines.

3.2.3 Search
The search for databases that might fit these criteria took place via Google and hub pages like Wikipedia, with links to both free and commercial Prolog/Datalog systems. The majority of these systems are simple standalone interpreters or embeddable interpreters in JAVA, without many features or documentation and at least lacking one of the criteria. This left a small list of databases that seemed to live up to the criteria or otherwise worthwhile to look into. These are mostly the bigger free systems. These databases will be discussed in section 3.2.5.
3.2.4 Quick overview of possible databases

<table>
<thead>
<tr>
<th>Name</th>
<th>JAVA API</th>
<th>Database</th>
<th>License</th>
<th>Stable</th>
<th>Easy</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRIS</td>
<td>Present</td>
<td>Semi-Present</td>
<td>LGPL</td>
<td>+</td>
<td>+</td>
<td>Does not support lists</td>
</tr>
<tr>
<td>SWI-Prolog</td>
<td>Present</td>
<td>Present</td>
<td>GPL+</td>
<td>+</td>
<td>+/-</td>
<td>Not portable</td>
</tr>
<tr>
<td>DLV System</td>
<td>Present</td>
<td>Present</td>
<td>GPL</td>
<td>+</td>
<td>+/-</td>
<td>Does not support lists, Uses an observer pattern</td>
</tr>
<tr>
<td>GNU Prolog</td>
<td>Present</td>
<td>Not present</td>
<td>LGPL</td>
<td>+/-</td>
<td>+/-</td>
<td></td>
</tr>
<tr>
<td>XSB + Interprolog</td>
<td>Present</td>
<td>Present</td>
<td>GPL</td>
<td>+/-</td>
<td>-</td>
<td>Hard to configure</td>
</tr>
</tbody>
</table>

3.2.5 Database details

**IRIS reasoner**

The IRIS reasoner is a Datalog database that has a user guide, API with Javadoc, and allows external databases to act as data storage. The database of the program itself is stored in-memory. For the purposes of this project it should only be used for the query that it is executing and not to store the whole database. An advantage of this reasoner is that it is a standalone library that does not need changes with administrator rights. Downside to the database connection is that the functions still need to be written as an extension to the existing in-memory storage to be able to retrieve the facts from the database.

**DLV System**
Link: [http://www.dlvsystem.com/](http://www.dlvsystem.com/)

DLV is also a standalone library that has a documented API, ODBC connections, and a start-up guide, but the code is harder to write than the IRIS code. It uses an observer pattern that allows for asynchronous calls to the Prolog interpreter, but the queries that will be written will very likely not be asynchronous. While using the program only one query at a time will be executed. This system could come in handy when developing heavier applications, but makes the code needlessly complex for this purpose by requiring the user to code and register different observers for just one query.

**SWI-Prolog**

SWI-Prolog is a Prolog environment that has a JAVA and ODBC interface and satisfies the requirements as middleware perfectly. It also seems well-maintained and the database connection is more integrated than in IRIS. The downside however is that the DLL that has to be called has to be in the system’s PATH variable. This means that the installer will need to have the right to put either the DLL in a system directory that is in the PATH by definition or have the right to change the PATH variable. These rights are currently not available and quite hard to obtain.
GNU Prolog on JAVA
This version of GNU Prolog is a Java library that only implements Prolog and no database connections. It is a library like IRIS and at first sight well-documented and explained. The installation and construction of the first terms is explained, but then it appears that constructing queries is done on a parser-like way: you construct a few atomic terms first and combine these terms into bigger terms and finally into a query, which is harder than the way IRIS works.

XSB with Interprolog
XSB is also a well-documented Prolog engine that supports ODBC through Prolog, but needs Interprolog for its JAVA interface, which has a stricter license. After installation, the XSB environment works, but the Interprolog environment has troubles finding the XSB executable. Only after contacting support, it was able to find the environment, but would still not compile all examples.

3.3 Conclusion
Keeping in mind the criteria we could choose more than one system and if time is unlimited they could probably all be configured to fulfil the needs of the project. Time, however, is short, so a system that is easy to work with is preferable. Using simple code will also reduce the chance of a crash or error. For this reason IRIS seems to be the best choice, as it has the same functionality as GNU Prolog, but IRIS is simpler and the absence of an integrated ODBC connection is less of a problem here: The IDataSource interface has to be implemented and the ‘getter’ overwritten to get a DataSource for a specific database. GNU Prolog does not handle Databases at all, which makes it much less suitable for the intended purposes. DLV’s observer pattern is needlessly complex for this project, but this is not as big a problem as not having database functionality, which causes DLV to stand above GNU, but below SWI and IRIS. SWI and XSB would not work, so these could not be fully judged on their functionality. SWI seems to have everything we want, but because it only can be configured to work while having administrator rights, it is considered second best. XSB is on the bottom of the list due to the (failing) two component structure that is needed.

Using IRIS will create a layer over the database that allows for Prolog querying and also for an easy change of databases, as only the get function has to be overwritten to be able to query the new database. This way, all databases that have as little as a simple function to read data can be used. RIP will use the Oracle database for this as it is the most used standard company database.

3.4 Follow up
The fact that some interpreters do not support list structures was discovered after the implementation of IRIS in the project had already started. Since SWI came in second and supports lists, the possibility of obtaining Administrator rights was discussed. The outcome was not to use SWI and that the combination of only recursive functions in Java and data storage in Oracle is favoured above a combination of Java, Oracle and a Prolog tier in-between. The reason for this is that RIP will probably be far more popular than foreseen and will have to be supported
by ASML’s ICT division. This division probably cannot program any Prolog and would rather not have an extra library to maintain.

The study indicates that there are very useful tools for this kind of data for future use, when RIP is scaled up to a larger type of environment. It would be useful to investigate the use of these tools again when the need for this upscale arises.
4 Approach

4.1 Introduction
This chapter describes the requirements specification process for RIP that produced the user requirements, software requirements and architecture design.

4.2 Requirements Specification Process
The project started with a main introduction in the problems ASML was facing. To elaborate on this an analysis of the task of the service engineers was made, to show what they had to do while repairing a machine and what the tool could do to shorten the time it takes. This task analysis is summarized in appendix A.

After this analysis, an initial set of requirements could be set up that was stored in the form of a User Requirements Document (URD), which includes textual requirements, use cases, and premature pictures of the visualisation of the data in the demo tool. Following this initial set of requirements, the URD has gone through several iterations with two of the stakeholders to get a complete set of requirements: Johannes Mulder, who leads this project, markets it to the employees at ASML and knows how the integrators and people at ASML in Veldhoven are going to use the tool, and Rene Fussenich, who knows what the people “in the field” (i.e. the service engineers) will expect this tool to do.

It was not feasible to follow an employee around to see how he would be able to use the tool for the reparation of a broken machine, because the machines do not break on command and many of them are stationed outside of the Netherlands. Luckily Rene and Johannes had a good idea of how a service engineer goes to work, so the tool will fulfil their needs. It will be necessary to conduct tests in the field with a more mature version of the tool than the to-be-delivered demo version though, before it is certain that the tool works properly.

The final set of user requirements (summarized in Appendix B) were translated to a set of software requirements (summarized in Appendix C). Finally, the translation to the architecture was made and in the Architecture Design Document (summarised in Appendix D).

4.3 Looking back
Due to the time constraints on this project, the coding had already started while the requirements were not in their final stage. In hindsight, working with unfinished requirements might have cost more time than finishing the requirements, as some details of how cables are connected were misunderstood and had to be re-implemented. On the other hand, the coding and reviewing of the prototype did help to better understand the initial requirements of the user and might also have saved more time then continuing to discuss the requirements. The ‘agile’ method might have worked better here as the coding of a small piece of work might have contributed to the understanding of the rest of the application while not having to re-implement functionality.
5 Design approach

5.1 Introduction
In this chapter, the general solution to the problem description is introduced in terms of the design approach to the software framework. It consists of a 3-tier architecture\(^2\) with the following layers: Database, Data and application logic, and Presentation. The Model-View-Controller\(^3\) pattern is used to decouple the data and the view. The decomposition in packages is done according to this pattern. The solutions that are made according to this design are described in-depth in the chapters following this one.

5.2 Structure of the design
The final solution to the problem consists of four main sub-solutions that each form a part of the total RIP project: Importing data, integrity checking of the data, the representation of the data in a model and the visualisation. These parts also happen in this order in RIP.

All of these parts should be able to work independently from each other, to achieve maximal flexibility. Of course data needs to be available before the integrity checking can deliver useful results, but after importing it once, the checks can be done independent of the import process and can all be executed as stand-alone checks.

The representation in a model should also be as loosely coupled to the database as possible, so it can be easily replaced by a simple local database on a ‘fabtop’ (a laptop without internet connection for use in factories). All recursive logic is thus moved from the queries to the Java code, to allow a simple database to cover the queries.

Also the visual representation was originally meant as a demo tool, so it should be decoupled from the events and the model as far as possible, to allow changes to the model and visuals without one part affecting other parts of the application.

These requirements have led to the conclusion that a 3-tier architecture with a Model-View-Controller system should be used as a decomposition pattern. The three tiers are the presentation layer, the application logic + data layer and the database. Compared to the MVC system, the presentation layer will act as the View. The database will be part of the model together with the application logic + data layer and only the events from the application logic will be part of the Controller model. The 3-tier architecture is normally used when the layers are divided over a separate Database Server, Application Server and a Workstation, but the division here is the same even though the application logic + data layer and the presentation layer are on the same machine. In the future, it is possible to separate the layers quite easily, because the MVC model was used and moved to an Application Server and Workstation, when desired.

\(^2\) [http://channukambalyal.tripod.com/NTierArchitecture.pdf](http://channukambalyal.tripod.com/NTierArchitecture.pdf), retrieved on 12-10-2011

\(^3\) [http://en.wikipedia.org/wiki/Model-view-controller](http://en.wikipedia.org/wiki/Model-view-controller)
The events in the Controller model are not all events that the view generates, but only those events that process data and update the view with that data. Events that only change the appearance of the view without affecting any data, are still in the view layer to prevent the overhead that placing them in the Controller would cause.

An overview of the implementation in classes will be given in chapter 7.3.
6 Importing

6.1 Introduction
A major part of the project evolved around the importation of several data sources into the database. In essence, this project started as a feasibility study to see if it is at all possible to construct the netlists (the maps) from these data sources. This seems to be possible indeed, but not all the needed data is provided in a ready-to-read format yet. Cable overviews can be read quite straightforwardly, but require some pre-processing before the cables and connectors can be entered into the database. TCE files are used for the length of the cables and are in a readable text format, but version control is needed to separate the different versions (and thus properties) of a cable. SAP files contain the same information and will not be used at this moment, but could be used later as a crosscheck file for the length. It is also possible to extract the signal and pinning information from the EDArch files and combine this with the knowledge about standard pinning and cables to construct the paths, but this is not implemented fully yet. Finding lengths of circuit boards in the EDArch files is something that still has to be analyzed, but the data and the possibilities to construct the required information are all available. For the purpose of giving demos of RIP, a test set containing already all the data is used as input.

In short: it is possible to construct usable netlists using these data sources, but for now only the test set is fully importable and the main data sources (TCE, EDArch, CO's and SAP) are described, but only partly implemented and thus not used for import yet.

6.2 Cable Overviews

Description
A Cable Overview is an Excel file in which each row contains a connection between: a connector on the first location, a cable in between, and a connector on the second location. Every column in the Excel sheet is a property of either one of the connectors or the cable, so importing could be done as easily as mapping every cell to a cell in the database. However, in this case the data needs some processing: The 11NC of each cable misses the first 4 numbers, so the standard “4022” prefix is added. The version number will be added by the database insertion function (see TeamCenter Engineering (TCE)). Also the module of the connectors is missing, and we cannot simply use the module of the cable for this, as this can be different from the module that the connector resides in.

Moreover, the Excel file should only be filled with values and not formulas like LOOKUP, which causes the file reader to fail. The cable overviews normally do not contain these formulas, but caution should be used.

After insertion of the cables and connectors, the connection will be made between the second connectors, so this cable will be a part of a path. This connection is red in Figure 6.1. Note that this circuit board is seen in RIP as a cable.

The mapping of columns to properties is done as described in table 6-1:
### Table 6-1 mapping of columns to properties

<table>
<thead>
<tr>
<th>Column</th>
<th>Property</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Location</td>
<td>Connector 1</td>
</tr>
<tr>
<td>B</td>
<td>Xnr</td>
<td>Connector 2</td>
</tr>
<tr>
<td>C</td>
<td>Type</td>
<td>Cable</td>
</tr>
<tr>
<td>D</td>
<td>Gender</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>Xnr</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Gender</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>SACA-id</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>11NC</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>SAP name</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Module</td>
<td></td>
</tr>
</tbody>
</table>

### The algorithm for importing cables and connectors

```plaintext
for (each row) {
    read the cable information
    add a line to the log file for each property that is missing or has the wrong format
    add "4022" to the 7NC to make it an 11NC

    read the first location information
    add a line to the log file for each property that is missing or has the wrong format
    add "4022" to the 7NC to make it an 11NC

    read the second location information
    add a line to the log file for each property that is missing or has the wrong format
    add "4022" to the 7NC to make it an 11NC

    check if connector1 and connector2 are not the same
    get the length of the cable
    insert the cable in the database
    insert the connectors in the database

    find a possible connection to another connector on the same location as the inserted
    connector and connect these connectors
    insert a connection between the first and second connector in the database
}
```

### Insert connectors

If the connector already exists in the database or has no location, the insertion is terminated. If it did not exist, it is inserted and connected to all connectors that are on the same interconnection component (i.e. have the same 12NC, XNR and location), but that are of a different gender. These connections are green in Figure 6-1.

Figure 6-1 Two sets of two connected connectors on both sides of a circuit board. RIP sees this circuit board as a cable.
This way all connectors will be connected exactly once to each other: Suppose that we have a connection between x and y. In the case that connector x is inserted first, there is no connector y yet. Once connector y is also inserted, there is a connector x and the connection is inserted. After this both x and y already exist and a connection will never be made again.

```sql
if( connector does not exist AND has a location){
    List of OldConnectors =
    "SELECT connectorId
     FROM Connector
     WHERE location=location
     AND twelvenc=twelveNc
     AND xnr=xnr
     AND NOT (gender=gender)"

    Insert new connector;
    for(each old connector){
        connect old connector to new connector;
    }
}
```

**Insert cables**
The insertion of a cable will be described in the chapter 6.4

### 6.3 Electronic Design Archive (EDArch)

**Description**
EDArch is an archive that contains the layout of boards and circuits and can be used to find the signals that run over such a board or to calculate the length of a board. The archive however does not contain any information on pinning or lengths from cables, so the pinning on cables has to be constructed out of the pinning information from the boards. In the history of ASML many systems have been used to archive this kind of information, but for the sake of clarity, this project only uses the system chosen by the client.

**Layout of the EDArch**
In the EDArch directory there is a subdirectory for each version of a part. This subdirectory has an 8NC, which is the 12NC minus the first four numbers. For each part the latest version (highest version number) will be used to extract the information from. In the folder structure of that subdirectory, the information will be in \wir\8NC_130_top.1 with the 8NC without points on the place of 8NC.

**Layout of the File**
The file itself is a simple text file with a header and different blocks for the different connectors. Each block has a header as illustrated in this example:
Here the rule starting with $M \text{asm\_conn\_sym:}$ indicates the start of a new connector. After this a rule starting with $I$ must follow. This contains an identity that is not used ($\$1I743$), a list of signals (WSXWTLCWPRES2OUT ... WSXSPMPURGETEMP4P) and a list of properties in the form of $X=Y$. The properties that are used are $\text{REFDES}$ that contains the Xnr and $\text{INFO}$ that contains a link to an intranet site about this connector.

After this block a list of pins will appear in the same order and quantity as the signals, as illustrated in this example:

```plaintext
API $\$1I743$ \text{asm\_conn\_sym:x\_10x20\_a}$ $\#A1$
API $\$1I743$ \text{asm\_conn\_sym:x\_10x20\_a}$ $\#A10$
API $\$1I743$ \text{asm\_conn\_sym:x\_10x20\_a}$ $\#A11$
```

Every rule starts with $\text{API}$, followed by the identity, the pin order and the pin number in the form of $\#=$pin number. The pin order indicates the index of connected signal in the list in the bloc header. So signal WSXWTLCWPRES2OUT will be connected to pin 1 which has pin number $A1$, GND to pin 2: $A10$ and so on. If a pin is connected to a signal that starts with a dollar sign, it means that the pin is not connected to a signal at all and can be ignored. In this example the pin with pin order number 16 would be connected to $\$1N1491$ and thus ignored.

If a signal appears more than once in the same block, the database will merge the pin numbers belonging to a signal. VP5V_1, for example, will have pin number $A13$, $A14$, $A15$ in the database. So in the internal structure of the database, each connector has one pin (with possibly a long number list) for each signal.

**Constructing the rest of the pinning**

In the database there is always at least one connector that contains signal information and from there the path of a signal can be reconstructed under the following conditions:

- If a cable is a SACA cable, the pinning is the same on both ends. If there is, for example, a signal SIGNAL1 on pin 1 on connector A, then there is a signal SIGNAL1 on pin 1 on connector B on the same cable.

- Sometimes a segment of a path contains more than one connector with signal information. These connectors can be used to do a crosscheck to check if the signal is indeed still on the same pin. If this is not the case, the information is not altered on both connectors. If these connectors shared a SACA cable an error is given in the log file.
- If a cable is not a SACA cable and there is no signal information available about the connectors on both ends, a note should be made in the log file.

When all the connectors and connections are put into the database, the algorithm for adding signal information will start. It will take all connectors with signal information from the database and put them in a queue. The algorithm will randomly take one of these connectors, say $X$. Based on the conditions described above, the algorithm will try to copy the signal information of $X$ to each connector that has a connection with $X$.

When the algorithm succeeds in copying signal information to a second connector, say $Y$, it will add $Y$ to the queue of connectors with signal information. This way the algorithm knows that it will also have to check all connections of $Y$ later on as $Y$ has signal information now. Connector $X$, however, will be removed from the queue when all his connections have been visited and the next connector with signal information will be chosen to repeat the steps above. Once the queue is empty, there are no more connectors left to copy signal information from and the algorithm will stop.

If all connectors would be indirectly connected to each other and if all cables would be SACA cables, we would need only one connector to copy the information to all the others. Connectors that are not indirectly connected to a connector with signal information and have no signal information themselves will never be visited by this algorithm. If a cable is not a SACA cable, the algorithm cannot add the signal information to the connector on the other side of the cable, and the algorithm might skip every connection after that SACA cable, which will cause a lot of missing signal information. For this issue, a solution still needs to be decided on.

**Algorithms for adding EDArch signals to the path**

First all files that contain a 7NC that can be found in the database are listed. For each connector the information is added.

**Algorithm for adding the EDArch signals**

1. List all the EDArch files that have a 7NC from a PCB.
2. Read the pins from the files with the highest versions
3. Add the signals to the rest of the path

**Algorithm for adding pins from an EDArch file**

1. Get all connectors for this board from the database
2. //remove a part of the header for a connector
3. Find a line starting with I $1743 asm_conn_sym:
4. Remove all characters up and until the first space after the ‘:’;
5. //find the properties of this connector
6. Loop through the blocks separated by spaces until you find one containing a ‘=’;
7. propertyList = the block containing the ‘=’;
8. Xnr = The string between REFDES and ‘ in the propertyList;
9. //process this connector if it is on the board
   If (this connector is on the board in the database){
      site = String between INFO and ‘ in the propertyList;
Add the site to the connector in the database.

//add all the signals without $ and their pins
signalLine = the block until the propertyList;
signal = first word of the signalLine;
while( signal is not empty ){
    pinLine = read next line from file
    if( signal does not contain '$' )
        Add this pin and signal to the pinList
    signal = next word of the signalLine;
}

//End of signalList, on to forming pairs of pins
get the standard pairs for this connector type
for(each pair that is available){
    Add the pair’s pinNumbers and signals as a node
    Remove the pair from the pinList
}
Add the rest of the pins as single nodes

The signals of the first connector of a random path of cables will be connected and the algorithm will move to the next available connector with signal information to do the same.

Algorithm to add signals to the path

    queueSignal = all connectors with signal information;
    while(queueSignal is not empty){
        Take a random connector with signal information;
        Add all signals to the connected connectors;
        Dequeue this connector;
    }

The algorithm will then traverse all connected connectors to try to copy signal information.

Algorithm for adding signals to all connected connectors

    For(each connector2 connected to connector){
        If(connector2 has signal information){
            Copy_signals(connector2, connector)
        }else if(connector has signal information){
            Copy_signals(connector, connector2)
        }
        Add connector2 to the queueSignal
    }

Now signal information will be copied if one connector has no signal information and both connectors are at the end of a shared SATA-cable or are of the same type at the same location. When both the connectors of one SATA cable have different information, an error will be produced. This will also happen when the cable is not SATA or the connectors are of a different type at the same location.

Algorithm for copying the signal from connector to connector2

    If(connector has signal information){
        If(cable is a shared SACA cable){
            If(connector2 has different signal information){
                Write error note to log
            }else if(connector2 has no signal information){
                Copy signal information to connector2
            }
        }else if(connectors do not share a cable, but are of the same type){
        }else if(cables do not share a cable, but are of the same type){
        }
    }else if(connectors have different type){
    }else if(cables do not share a cable, but are of the same type){
        Copy signal information to connector2
Copy signal information to connector2
} else{
 Write error note to log
}

**Inserting components into the database**

The node will be inserted into the database with a couple of (pin, node) pairs.

**Algorithm for inserting a node into the database**

1. Insert the node in the database
2. **If** (the signal names of the pinList are not already on the connector){
   1. **for** (each pin in the pinList){
      1. Insert the pin as belonging to this node;
   }

The pin will be inserted when the signal is not already on the node, or else add this pin’s number to the pin already containing the signal.

**Algorithm for Inserting a pin**

1. **if** (the number is not empty){
   1. **if** (the signal of this pin is already on the node){
      1. **if** (the number is not already on the node){
         1. Add the number to this signal on the node
      }
   } else{
      1. Add the pin to the database
   }

---

**6.4 TeamCenter Engineering (TCE)**

**Description**

TCE is a database for all kinds of documentation within ASML that stores, among others, information about the cables. The information that is needed for RIP can be queried and exported to text files, so it can be easily imported into the database. In a later stage it might be possible to connect to the TCE storage system itself, so the database can be automatically kept up-to-date, but for this project speed and easiness of connecting to the data is given priority over up-to-date data, as this data probably will not change much.

Each file in the TCE directory has the following filename: *TCE-xxxx.xxx.xxxxx.txt*, with the 12NC on the place of the x’s. This way each file belongs to a unique cable, but much like the EDArch directory, the list of TCE-files will often contain multiple versions of the same cable. In this case, however, all the versions will be stored in the database as the version of the cable can affect the path of a signal. For now, only a change in the length of the cable is of importance, since this can influence the measurement of the TDR tool, but also the other properties of an older version of the cable are stored.

The list of TCE files is not read to the database at once, but is consulted every time a cable is inserted into the database. When this cable is not present in the database, the insertion
algorithm will search the TCE directory for all files that contain the first 11 digits of the 12NC, to find all versions of the cable, which can lead to three cases of insertion:

1. The algorithm does not find any files and one cable is added to the Cable table with a question mark as version number.

2. Only one file is found and this version is added to the Cable table, with the indication that there are no other versions.

3. Multiple files are found and a cable is added to the Cable table with the highest version as version number, with the indication that there are multiple (older) versions. Each older version is then stored in the OldCableVersions table.

**Layout of a TCE file**
A TCE file is laid out as in the following example:

<table>
<thead>
<tr>
<th>12 NC</th>
<th>@Item Description</th>
<th>@Quantity</th>
<th>@Sequence No.</th>
<th>@Item Rev</th>
</tr>
</thead>
<tbody>
<tr>
<td>4022.470.90401</td>
<td>@ATRS LNS LOAD X CABLE ASSY</td>
<td>@1</td>
<td>@</td>
<td>@RELEASED</td>
</tr>
<tr>
<td>4022.438.39575</td>
<td>@PROTECTIVE LABEL 20.3X12.5MM</td>
<td>@3</td>
<td>@3</td>
<td>@RELEASED</td>
</tr>
<tr>
<td>4022.438.33239</td>
<td>@5P 0BRND PIN STR CRP CBL G .7</td>
<td>@1</td>
<td>@6</td>
<td>@RELEASED</td>
</tr>
</tbody>
</table>

It starts with the 12NC of a part, which is different from the 12NC of the cable, as the cable is made up of different parts. The next column describes the part and the third column the quantity. To find the length the cable needs to be found within this part list and read its quantity. This will be a decimal number giving the length in meters. The difficulty is that a cable is not only composed of the cable itself, but also includes connectors, labels, etcetera. Moreover, the 12NC of the actual cable is not the same 12NC that the composed cable (and thus the TCE file) has. Another problem is that the cable description neither has a unique keyword nor a property to determine what exactly the cable is. Solving this issue was outside the scope of this project, so for this part a data analysis still has to be performed to find any patterns.

To determine which part is the actual cable, the algorithm looks at the parts that have a length (a decimal number as quantity) and will indicate the part with the highest length as the cable, since the other parts will mostly be connectors.

**Algorithm for finding a cable**

```python
if (there is no cable with the same 11NC and there is a 12NC){
    Find all possible versions in the TCE files with the same 11NC.
    if (no files are found){
        add an ? on the place of the version number
    }else{
        read the length of the highest version
        add the highest version in Cable
        if (more than 1 version is found)
            set the multiple versions field to true
        else
            set the multiple versions field to false
    }
}

for (all older versions){
    read the length of the cable
    Insert the older version cable in OldCableVersions
}
```
6.5  SAP

Description

SAP is a workflow management system, but also acts as a database. Among other functionalities, it stores and exports information about cables. This information (which is supposed to be in text file format) could be extracted by the import algorithm in a future project, so it can be used to crosscheck the length of a cable with the length from the TCE files. This project, however, will not implement any SAP import or check functionality, since the same information is already available from the TCE files.

6.6  Test set

Description

The test set is basically an extended cable overview with all the information that the tool needs (except for the SACA-Id of a cable). It also uses formulas such as LOOKUP to look up the pinning in the set of standard pinning pairs, which should be converted to values beforehand. This can be done easily by copying all fields and using the special paste option ‘paste values’ to paste them on the same place. Besides the cable segment, each row contains a signal and return signal that go over the segment, their pinning, and the length of the cable. This way almost every cable has multiple rows in the test set, as each signal running through it requires its own row. A set of multiple signals for the same cable is illustrated in table 6-2:

<table>
<thead>
<tr>
<th>Signal</th>
<th>ReturnSignal</th>
<th>Loc. 1</th>
<th>12nc</th>
<th>xnr</th>
<th>pin</th>
<th>pin</th>
<th>Cable</th>
<th>12nc</th>
<th>Loc. 2</th>
<th>12nc</th>
<th>xnr</th>
<th>pin</th>
<th>pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATWSRTISVP</td>
<td>ATWSRTISGND</td>
<td>ESCP</td>
<td>.636.3226</td>
<td>X6</td>
<td>2,3</td>
<td>7,8</td>
<td>AA TIS CHCK2 ESCP</td>
<td>.636.0873</td>
<td>UCB</td>
<td>.640.1150</td>
<td>X13</td>
<td>2,3</td>
<td>7,8</td>
</tr>
<tr>
<td>ISDATASPRP</td>
<td>ISDATASPRN</td>
<td>WLCP</td>
<td>.636.3382</td>
<td>X1-5</td>
<td>4</td>
<td>9</td>
<td>AA TIS CHCK2 ESCP</td>
<td>.636.0873</td>
<td>UCB</td>
<td>.640.1150</td>
<td>X13</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>ISWSRTISA</td>
<td>ISWSRTISB</td>
<td>WLCP</td>
<td>.636.3382</td>
<td>X1-5</td>
<td>1</td>
<td>6</td>
<td>AA TIS CHCK2 ESCP</td>
<td>.636.0873</td>
<td>UCB</td>
<td>.640.1150</td>
<td>X13</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 6-2 Multiple signals on one cable in the test set

It should be noted that not all columns have been included here for readability. A set of green or yellow cells following each other is a connector. The left side blue column contains the signals and the central blue column contains the cables. Furthermore, the rows in the test set are ordered per signal name, in the order of the path the signal follows, so it is visible at once how a path runs through the machine. The connector on the right side should be the same as the connector on the left side on the row beneath. In table 6-3, all connectors on the left are males and on the right females:

<table>
<thead>
<tr>
<th>Signal</th>
<th>ReturnSignal</th>
<th>Loc. 1</th>
<th>12nc</th>
<th>xnr</th>
<th>pin</th>
<th>pin</th>
<th>Cable</th>
<th>12nc</th>
<th>Loc. 2</th>
<th>12nc</th>
<th>xnr</th>
<th>pin</th>
<th>pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATWSRTISVP</td>
<td>ATWSRTISGND</td>
<td>MMCR</td>
<td>.472.9058</td>
<td>S22X2</td>
<td>2,3</td>
<td>7,8</td>
<td>IM WS1 SPOT IMCR</td>
<td>.636.0252</td>
<td>ESCP</td>
<td>.636.3226</td>
<td>X6</td>
<td>2,3</td>
<td>7,8</td>
</tr>
<tr>
<td>ATWSRTISVP</td>
<td>ATWSRTISGND</td>
<td>ESCP</td>
<td>.636.3226</td>
<td>X6</td>
<td>2,3</td>
<td>7,8</td>
<td>IM WS1 SPOT WLCP CBL</td>
<td>.636.0729</td>
<td>UCB</td>
<td>.640.1150</td>
<td>X13</td>
<td>2,3</td>
<td>7,8</td>
</tr>
<tr>
<td>ATWSRTISVP</td>
<td>ATWSRTISGND</td>
<td>UCB</td>
<td>.640.1150</td>
<td>X13</td>
<td>2,3</td>
<td>7,8</td>
<td>AA TIS CHCK2 ESCP</td>
<td>.636.1857</td>
<td>UCB</td>
<td>.640.1150</td>
<td>X10</td>
<td>2,3</td>
<td>7,8</td>
</tr>
</tbody>
</table>

Table 6-3 A path of three connected segments in the test set

With this test set, all information that is needed by the tool, except for the SACA-Id, is available in one file, so an algorithm could directly construct a path for every signal in the database.

The algorithm that uses this set has to be more generic though, so it can be re-used for all other cable overviews that do not contain extra information or an ordering. Thus it first imports the
data of cables and connectors from the set as it would do with a cable overview and does not use the order for connecting connections. While each cable is inserted, the length is read from TCE files as described before, but if those are not available the length from the test set is used. This is followed by another walk through the list of connectors: for each connector the test set is scanned again to find its pinning + signal combinations and convert those to its nodes.

This way, the paths that will be produced later on by using this import algorithm can be crosschecked against the test set to scan for mistakes in the import or path finding algorithms.
7 Representation of the data

7.1 Introduction
In this chapter, the structures for the database and application to store data are described. Both are based on the hierarchy within a machine as the basis and the rest of the data grouped in loose categories built around that. The structure for the classes of the application is based on the MVC pattern described earlier in chapter 5.

7.2 Database structure

Storing the data
The database structure for saving all the data is based on the natural relations between the objects in the machine. These objects can be divided in different parts:

- Physical objects, which are all the objects that are physically present in the machine, and consist of:
  - Modules: On the highest level we have the different modules in the machine (as described in the context). A machine is configured by combining a specific set of these modules. Each module itself is in fact a container of the cables and connectors and has no specific properties.
  - Cables and Connectors: These are used to make paths for the signals through the machine. Each cable has at least one and mostly two connectors with which they can connect to each other. A physical path will consist of a set of cables that are connected via their connectors. The location where two connectors are connected will be called an interconnection location, which can be a direct connection of two connectors or a connection via a simple circuit board or a hub. A connector is either a male or a female. Cables sometimes have different versions that can differ in length. This is important information for the service engineer, so all cable versions should be in the database. These are stored in the OldVersionCable table which saves older versions of the cables in the Cable table. The cableId or the combination of 12NC, SAP name, and module can be used to connect the highest version cable to the older versions. The rest of the information can differ.
  - Segments, Nodes and Pins: Each connector consists of a number of pins that are each connected to a single wire within a cable. A cable connects two or more connectors on each side to each other. Electric signals that travel over these pins usually come in pairs, so they need to be saved in pairs as a node. A node does not have to contain a pair of pins, it can also contain only one pin or even have more than two, but then we know for certain the amount of return signals a signal has. We use a StandardPinningPair table with hardcoded standard pairs of pins for connector types to be able to form these nodes. A node connected to a wire connected to a node is called a segment, but a segment is easily deducible from the information of a Node, its connector and cable and the signals, so it is not stored separately.
- Invisible objects that are present in the machine, but cannot be categorised as physically touchable consist of:
  
  o The electrical signals that run through the machine. They will each be tied to the set of pins that they run over in the form of giving each pin a signal that it is tied to.
  
  o The connections between the connectors, which are stored as the edges in undirected and directed graph form. This split will be explained later in this chapter.

- Objects that are not in the machine, but applicable to them consist of:
  
  o Exceptions that apply to the physical objects, except for the modules.
  
  o Sources outside the application that contain information about a specific object.

Also the data needed for the application itself, which is not tied to the machine or the signal paths, is saved. This consists of the Users and the Paths to data sources.

Some of the tables in the database have an ID field that is not the primary key of the database. These IDs are used as an index to speed up the database queries, because filtering on one field is easier and faster than filtering on a combination of three fields. Their uniqueness is enforced through an autonumbering sequence for every ID field.

**Storing connections in particular**

Those connectors have connections to each other, comparable to edges in a graph. Each edge is stored as directed in `ConnectorToConnector (CTC)` and as undirected in `UndirectedConnectorToConnector (UCTC)`.

<table>
<thead>
<tr>
<th>UndirectedConnectorToConnector</th>
<th>ConnectorToConnector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connector1</td>
<td>Connector2</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
</tr>
</tbody>
</table>

Table 7-1 This table stores connections from one connector to another in directed form

<table>
<thead>
<tr>
<th>ConnectorToConnector</th>
<th>Connector1</th>
<th>Connector2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

Table 7-2 This table stores connections from one connector to another in undirected form
T prevent transposed duplicates like (A,B) and (B,A) in the results we use CTC. The disadvantage is that we have to search both rows instead of only the first for all connections to one connector. The advantage of the undirected version is that all connections can be found from the first row. The obvious disadvantage is that twice the storage space is needed. Since relatively more storage space than CPU power is available, the decision was made to have both the CTC as well as the UCTC table. Next follows an example based on the problem of spider cables (Chapter 8.2) to illustrate the advantages of also using UCTC.

We have twice the table CTC: CTC1 and CTC2 and four columns CTC1.C1, CTC1.C2, CTC2.C1, CTC2.C2. To check for two connections to the same connector we need to check if the connector in column one of CTC1 is the same as the connector in column one of CTC2 and that they are not the same connection (column two of CTC1 is not column two of CTC2) and that they are not per exception forming a loop (so column one of CTC1 is not column two of CTC1, which immediately implies the same for CTC2). The problem is that if we only check the first column, we miss the connections that start from column 2, so the query should be done again for column two. The same is true for trying to find the spider connection in CTC2: both columns should be checked for the starting connector, so we have four different queries in total.

<table>
<thead>
<tr>
<th>CTC1.c1 = CTC2.c1</th>
<th>CTC1.c2 = CTC2.c2</th>
<th>CTC1.c1 =/= CTC2.c1</th>
<th>CTC1.c2 = CTC2.c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC1.c2 =/= CTC2.c2</td>
<td>CTC1.c1 =/= CTC2.c1</td>
<td>CTC1.c2 =/= CTC2.c2</td>
<td>CTC1.c1 =/= CTC2.c2</td>
</tr>
</tbody>
</table>

In the undirected table all connections are guaranteed to be available by only searching for the connector in the first column, and then preventing all loops that are in the results, because of the insertion of the transposed duplicates. This is already done in the other query anyway for safety reasons, so this reduces the four queries needed with CTC to just the first query, when using UCTC.

**Backup**

The ability to back up the database is limited in RIP. It can store precisely one backup of the current database that is visible in figure 7-1. This backup does not include the tables that do not change during import: Path, RipUser and StandardPinningPair. At this moment, every time the user presses the import button, a backup of the database is stored in the same table structure but with 'Backup' in front of each table name. The previous backup is deleted by this operation.

This way of making backups is the simplest and safest way and was chosen because the focus was not on back up functionality, but on providing a simple possibility to backup the database in case something goes wrong during import. The only way to restore a backup is to let an Administrator copy all the values of the backup tables back into the real tables, which should be easy as “TRUNCATE Table; SELECT * FROM BackupTable INTO Table;”, as they have the same structure.

A better way would be to store the complete database in backup files in binary or SQL format, which allows for easy distribution of a database and more backup versions, but also needs a safe location to store this sensitive data.
7.3 The data model

Storing data in the application

In the application we use the same type of natural hierarchy as before with the database, but the classes that take over the role of the tables can now be grouped into categories that make the division between the different kinds of functionality more clear. The relations between these categories are also visible are visible in the schematic class model in figures 7-2 and 7-3. A more detailed view of what each class is intended to do can be found in the Javadoc description.

We will group the objects needed to build a path (Invisible and Physical objects) into the category ‘physical’. Also the related objects Exceptions and Outside Sources will be mixed through the properties of the ‘physical’ components as Strings and not as separate objects. This part will be the basis of the data model.

The rest of the classes that we use for our software framework are grouped on basis of their
function. The logic for importing the data into the database is grouped in ‘import’ and the logic to query the database is grouped in ‘database’ and ‘query’. Query here only contains the functionality to query the data from the database. The logic to insert and update items in the database are grouped in ‘database’. This division was made at the time that Prolog was still an option for querying the database and thus needed a different class, using a Prolog data source. The simple update and insert functionality was much easier to implement in the database’s own pure SQL, so it was put in the database class as insertion and update functions. All these four categories together form the data model of the Model-View-Controller construction. All classes that take care of the user interface are grouped in the ‘application’ category. The controller is a bit of a lonely class, but it handles all events that need both the model and the view. It thus stands above them and needs a separate class. As it handles the complete application and the main class of the application (RIP) only has the function to start this controller, it is grouped together with the main class in the package ‘rip’ that contains the functionality that lets the user interact with RIP.

Now all main functionality is described, but there are still some classes left that contain handy functions, like rounding and pair types, and the global enums and constants to provide consistency throughout the code. These classes are not really an explicit part of either the data model, or the controller, or view as they are used in all of them, so they are grouped in the ‘util’ category which is the standard name for a package containing such functions. It is drawn separate from the schematic class model, as it is used throughout the whole code and the dependencies would make the class model hard to read.
Figure 7-3: The division in packages and classes of RIP

Figure 7-2 The util package
8 Data integrity and exceptions

8.1 Introduction

Description
The data that is imported from sources that are made manually will contain errors like missing data, typographical errors, inconsistencies within the data, and so on. Also some data is not wrong, but still inconsistent with the standards. To work properly RIP should check for known errors and try to handle exceptions to the standards as well as possible. In this chapter it is described what the known issues are and how they are handled and when.

What check is done when
The issues are handled in the following order:

Import time:
- Finding the multiple versions of cables
- Finding missing properties that can cause a breach in the path
- Use standard pinning pairs for forming nodes

Voluntary checked by pressing the corresponding button:
- Finding mismatched gender pairs
- Finding 12NC changes
- Finding cables with only one connector
- Finding spider cables that branch on a signal

Visualisation time:
- Highlight spider cables
- Add an exception for a possible signal change
- Highlight multiple pins for a signal
- Highlight a breach in the path
- Add an exception for a cable with only one connector
- Highlight cables with multiple versions

8.2 Spider cables

Description
Spider cables are cables that consist of one connector on one side and multiple connectors on the other side of one cable. As far as this project is concerned there are no other forms, like multiple connectors on both sides. A spider cable can be both an error in the data or a normal form of cable. An example of an error is that there should be two cables with two connectors, but due to a copy paste error, one of the connectors is now the same as the connector on the other cable. An example of a normal form is shown in figure 8-2.
Each spider cable can be put in one of two categories:

- Spider cables that solely show a branch in the cable but not in the path of a signal. Each signal occurs once on each side of the cable and the branch is not noticeable in its path (The red line is the signal) as shown in figure 8-2

- Spider cables that show a branch in both the cable and the path of a single signal. The signal path will have a node on the left-side connector which is connected to a node on both the right-side connectors. (The red line is the signal) as shown in figure 8-1 and 8-3.

The first category is not a big problem as the construction of the path of a signal is not influenced by this cable. It can, however, influence the measurements of the length of the cable, so a warning has to be issued if a segment is part of a spider cable. The fact that a cable is a spider cable can only be inferred from the database. This can be checked once for every map, or an extra field could be made in the database to indicate if a cable is a spider cable. RIP will create a view to get the spider cables from the Cable table so those can be checked against the cables on the path.

The second category of spider cables is a bigger problem, since they will affect the path of a signal. This issue will be addressed in the application while creating segments from all the nodes in a path. A branch will start with a node on the left-side connector and end with more than one node on the right-side connectors. It is also possible to find all spider cables by an Oracle query

**Oracle Query**

Select all the cables that have a connector C1 that is connected to a connector C2 and a connector C3, where C1 =/= C2 and C2 =/= C3, but do have all the same cable.

```sql
CREATE VIEW SpiderCableView AS
SELECT DISTINCT CB1.*
FROM UndirectedConnectorToConnector CTC1,
UndirectedConnectorToConnector CTC2,
Connector C1, Connector C2, Connector C3, cable CB1
WHERE CTC1.connectorId1 = CTC2.connectorId1
AND NOT (CTC1.connectorId2 = CTC2.connectorId2)
AND NOT (CTC1.connectorId3 = CTC2.connectorId3)
AND C3.connectorId = CTC2.connectorId2
AND C1.connectorId = CTC1.connectorId1
AND C2.connectorId = CTC1.connectorId2
AND C1.cableId = C2.cableId
AND C1.cableId = C3.cableId
AND C1.cableId = CB1.cableId
```

Figure 8-2 A spider cable, not branching on signals

Figure 8-3 A spider cable, branching on signals
Algorithm for finding spider cables (branched on signal)
The spider cable with one connector on top and two on the bottom is illustrated schematically in figure 8-4. From each connector only the applicable node is taken.

```java
   topNode = get a topNode
   segment = new Segment();
   segment.topNode = topNode;
   bottomNodes = get all bottomNodes
   If bottomNodes.length() > 0{
       topNode is part of a spider cable
       each bottom node is part of a spider cable
```

8.3 Connector signal change

Description
When the path of a signal ends, it is possible that the signal has not ended, but that its name has changed. The path that is shown is not complete in that case. The name changes do not follow a solid pattern, so we can only detect the fact that this path physically runs further, by checking if there are still any connectors connected to the start and end of the path. An example of this problem is shown in Figure 8-5, where the red and blue paths have different names, but should both belong to the same signal path.

To detect this, we start by collecting the set of connectors that have a connection to the end connector or start connector of the path. All connectors that have the current signal running through them will be removed from this set, so we have a set left of connectors that exclude the current signal, but are connected to the start and end connector. Each of those connectors will have this 'connector signal change' exception as a warning to the user that the signal could be continuing. These exceptions are added right after the signal path is constructed in the data model.
Query

SELECT DISTINCT
C.connectorId, C.xnr, C.location, C.gender
FROM
UndirectedConnectorToConnector CTC, Connector
WHERE
CTC.connectorId1 = connectorId
AND C.connectorId = CTC.connectorId2
AND NOT EXISTS(
    SELECT PN.signalName
    FROM PinNode PN, Node
    WHERE PN.nodeId = N.nodeId
    AND N.connectorId = CTC.connectorId2
    AND PN.signalName = 'signalName'
)

8.4 Multiple pins for a signal

Description

There can be multiple pins on one node that are used for the same signal. At the time of insertion, it will be checked if a pin with the same signal already exists on its node and if it does, the pin number will be concatenated to the existing pin numbers for this signal, separated by commas. So a connector that has four pins with the same signal, say pin 1, 2, 3 and 4, will be saved as a single pin with the number ‘1, 2, 3, 4’. Highlighting this in the map can be done by a simple line of code that checks if the pin number contains a comma as shown in figure 8-6.

Oracle Query

//Insertion
SELECT pinNumber
FROM PinNode
WHERE nodeId=nodeId
AND signalName='signalName'

If there are results
UPDATE PinNode
SET pinNumber ='results.pinNumber, pinNumber'
WHERE nodeId=nodeId
AND signalName='signalName'

Else just insert the pinNumber and signalName.

INSERT INTO PinNode(pinNumber, signalName, nodeId)
VALUES ('pinNumber', 'signalName', 'nodeId')

Algorithm for highlighting the pin number

if( the pin number contains a (",")){
    highlight the pin number on the map
}
8.5 Non-standard pinning

**Description**

For every connector of a standard type, the pairs that the pins form are in a standard order. For example: on the D-sub 15 connector the pinning is as stated in the table 8-1.

For the test set, the pinning from that test file is used, but for the conversion of EArch files to nodes, the standard pinning is used to figure out which pins form pairs. The problem is that there are connectors that are not in the list of standard pinning, so there is no way to tell which pins are connected.

A connector could have a single loose pin, like pin 15 on the D-sub 15, but also three pins that should form a trio in a three-phase connector.

The database can handle all numbers of pins that are connected to the node, but the data model still queries only loose pins and pairs of pins from the database, because no solution to make nodes of three pins is implemented yet.

<table>
<thead>
<tr>
<th>Pin 1</th>
<th>Pin 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 8-1 The standard pinning of a D-sub 15 connector

8.6 Mismatched Genders of pairs

**Description**

A male connector can only be connected to a female connector. If a male connector is connected to a male connector or a female connector to a female connector, there is a mismatch in the source data. These connections will be found by searching for couples of connectors that should be connected, but are not connected, because of their equal gender. The criteria for connection are twofold. Firstly, the connectors (I think this should be plural) have to be the same (i.e. share the same location and Xnr), but their gender should differ. Secondly, the connectors should be connected by a signal path (i.e. share a signal name), but should have a different cable (so that it is not a branch). If these criteria are met for two connectors, but they are not connected in the database (i.e. the connection is not in UndirectedConnectorToConnector) and their gender is the same, then these connectors will be added to the list of gender mismatched pairs.

**Oracle Query**

```sql
SELECT DISTINCT
PN1.signalName, C1.gender, C1.location, C1.Xnr, CB1.sapName,
CB2.sapName
FROM
```
PinNode PN1, PinNode PN2, Node N1, Node N2, Connector C1, Connector C2, UndirectedConnectorToConnector CTC, Cable Cb1, Cb2

WHERE
PN1.signalName = PN2.signalName
AND N1.nodeId = PN1.nodeId
AND C1.connectorId = N1.connectorId
AND C1.cableId = CB1.cableId
AND N2.nodeId = PN2.nodeId
AND C2.connectorId = N2.connectorId
AND C2.cableId = CB2.cableId
AND C1.location = C2.location
AND C1.xnr = C2.xnr
AND C1.gender = C2.gender
AND NOT (C1.cableId = C2.cableId)
AND NOT (CTC.connectorId1 = C1.connectorId
AND CTC.connectorId2 = C2.connectorId)

8.7 Breach in path

Description
Each signal that is inserted into the Path constructor should generate exactly one complete path. If a signal generates multiple paths, then a connection is missing between two connectors that should be connected. This can have multiple causes, but it is always an indication of one or more errors in the source data. In RIP, a breach will be indicated by a red bar on the path that states that “The path is broken here” as exemplified in figure 8-9. In the data, the breach can be detected by the presence of multiple paths.

Phenomena that are known to cause the broken path exception are:

- Mismatched genders, where two connectors that are male and female are disconnected, because they both have the same gender in the source.
- Mismatched XnrS, where a male and female connector on the same location should be connected, but are not, because the Xnr is not the same on both sides. This can be an error in the data or done on purpose.
• An 11NC that is different on 2 instances of the same location, where again two connectors on the same location should be connected, but are not, because now the 11NC of the location is not the same on both connectors.

• Discontinuity between cable overviews, where a cable runs through multiple cable overviews, but one or more cable overviews are missing rows of data that are part of this cable. The quality of the cable overviews is known to vary between projects.

This situation could also be caused by exceptions that are undetectable or unforeseen like typographical errors in the data.

Each breach should be examined manually by locating the connector before the breach in the source document and by trying to find out what causes the missing connection.

**Algorithm for finding a broken path**

```java
for (int i = 0; i < segmentViewTrees.size(); i++) {
    Make tree i into a path
    if (!(i == segmentViewTrees.size() - 1)) {
        // there is a next path so here is a breach;
        Add a broken path banner
    }
}
```

8.8 A cable with one connector

**Description**

Sometimes a cable ends prematurely, which means that it has a starting connector but no end connector. This can be normal behaviour if the cable is in reality a circuit board that also acts as an end component, like Figure 8-10, but it could also be an error if there is supposed to be a connector at the end. This is always the bottom node, as the path making starts with adding a top node. RIP adds an exception to the top node of a segment that warns the user that it could be an error that no bottom node is present.

**Algorithm for finding cable with one connector**

This is a part of the path creation algorithm. It returns the incomplete segment with an exception when the top node of a segment has no bottom node.

```java
For (each segment) {
    Node topNode = “get a the top node for the segment”;
    if (!topNode.hasChildren()) {// no ending node
        topNode.addException(“No end node”);
        return topNode; // return the node without connecting any end node
    }
}
```

Figure 8-10 A board-cable combination that is seen as 1 cable
8.9 Multiple versions of cables

Description
Often a cable has more than one version. The last number of the 12NC denotes this version. If there is only one version that number will be 1, the next version will be 2, etc. This way of numbering limits the maximum number of versions to 9. The cable length can change per version, so these versions needs to be detected. This is done while importing the cables. Upon trying to insert a new cable in the database, the algorithm will check the version numbers by searching for TCE files on the TCE path that contain the 11NC (the 12NC without version number). The returned list of files then contains all the different versions. Searching for 4022.470.7287, for example, returns 2 versions:
- TCE-4022.470.72871.txt
- TCE-4022.470.72872.txt

In the database, the highest version is inserted as the cable that is part of this module, with the indication that there are other cable versions by setting the multipleCableVersions field to ‘Y’.

The other versions are inserted in the OldVersionCable table and can be recalled when necessary. If a cable has only one version (even if this is not version 1), there will be no old cable versions and the multipleCableVersions field will be set to ‘N’.

If the algorithm cannot find any file that contains the 11NC, it will insert only 1 version of the cable with a question mark as the last digit.

Every time the algorithm comes across this 11NC, it will now take the highest version of the cable from the database and use it as the cable for that segment.

If the 11NC is missing it will insert an empty field and use the SAP name of the cable to find the same cable. It is assumed that these cases will be rare and that the combination of no 11NC and this SAP name will be unique.

When constructing a path the path constructing algorithm will request a list of other versions of the same cable that have a length different from its own length. If there are any other versions with another length, the pairs of the other version’s 12NC and length will be added to the properties of the cable.

SQL query for finding existing cables
```
SELECT 
cableId 
FROM 
Cable 
WHERE 
sapName = 'sapName' 
AND twelveNc LIKE 'elevenNc_'
```
**SQL query for finding other versions with another length**

```
SELECT
twelveNc, length
FROM
OldCableVersion o, Cable c
WHERE
o.twelveNc LIKE 'twelveNc.substring(1,11)_'
AND c.twelveNc='twelveNc'
AND NOT (o.length = c.length)
```

**Algorithm for inserting cables**

```java
int cableId = getInt("Search for existing cable");

if(cableId == -1){
    //no existing cable found, so search for all the versions of cables
    String[] files = All files starting with TCE-elevenNc;
    Arrays.sort(files);  //sort on version number
    multipleVersions = 'N';  //standard no multiple versions
    if(files.length == 0){
        //no cables found so add an ? as last digit
        twelveNc = elevenNc + ?;
    }
    else{
        //use highest version
        twelveNc = files[length-1];
        if(files.length > 1){
            multipleVersions = 'Y'
        }
    }
}

cableId = insert cable(twelveNc, multipleVersions);

//Insert the children
    for(int i = 0; i < files.length - 1; i++){
        Insert old version cable with twelveNc = files[i]
    }

return cableId;
```
9 Application logic

9.1 Introduction
This chapter describes the functionalities that are not part of the import process, data integrity or the visualisation, but provide the logic behind the other actions the user can do from the user interface.

9.2 Searching
There are two standard search functions: one to search for signals and one for physical components in the database. These two functions have been split for clarity: the results of signals need to be treated differently from the results of physical components. In both searching algorithms, the wildcard ‘*’ can be used as ‘any string of characters’ and is replaced by the ‘%’ in the oracle query. Wildcards at the beginning and the end are automatically added, so a user always gets all results that have a part of the query in them. For signals, all results with a part of the query in the signal name will be returned. For physical components, all parts with a part of the query in either their 12NC or SAP name will be returned. Since the DISTINCT keyword is used, results will not be returned twice when both the 12NC and SAP name contain the same query.

9.3 Path finding
The path in RIP is the complete set of segments that belongs to a signal and (almost always) a return signal. This path, as do most paths, consists out of a set of edges and points, which are formed by respectively the connections and connectors, as described in the introduction of this paper.

In a path there are two kinds of connections between the connectors:

- The connection between two connectors that are on the same cable.
- The connection between two connectors that connects two cables to each other via an interconnection component.

![Figure 9-1](image.png) The two kinds of connections between connectors. Each blue-orange row is a segment of a Cable Overview. Blue is a connector, orange a cable.
Both types of connections are already made at the time of import, so the path itself is in fact also established at that time. The path making algorithm does not change the path itself, but merely changes the form in which it is represented. It, however, does try to detect abnormalities on the path like a spider cable or the presence of a break in the path and will act correspondingly when it has found one. To understand why a certain collection of data delivers these errors, one would need to look at the import algorithm and the data source.

The path algorithm uses an SQL query to find all the connections and will form this into a path in the format that is in the data model. This code will distinguish between connections on basis of the cable: If two connectors do not share a cable, it is a connection via an interconnection component; else it is a connection via a cable.

Each row in the SQL query results contains, besides the connection, all data about the two nodes on the connection, the pins, the two connectors and the cable. The query algorithm delivers this result in a two-dimensional array in which each row is an array containing one record, where each array cell contains a result cell in the following order, where cell 1 is the id of Node 1, cell 3 the id of Connector 1, etc.:

<table>
<thead>
<tr>
<th>Node1</th>
<th>Node2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connector1</th>
<th>Pin 1</th>
<th>Return pin 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>12nc</td>
<td>Xnr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connector2</th>
<th>Pin 2</th>
<th>Return pin 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>12nc</td>
<td>Xnr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cable1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cable2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 9-2 The fields (arranged by component) that are returned by the query for the path of a signal

The algorithm that will make a path out of this data will iterate over the results and turn them into one or more trees until there are no results left.

As the rows in the results are in arbitrary order, the algorithm will select an initial row and turn it into a tree with two points (the 2 nodes) and 1 edge (the child-parent connection). This edge can represent either a cable or an interconnection location. If it is a cable, both nodes will share
the same cable; if it is an interconnection location, both nodes will have a different cable. The row is then removed from the array with results.

After this, the algorithm will loop through the remaining results until it finds a connection that matches one of the nodes in the tree. It will then connect the matched node in the tree to the connected node from the results. In almost all cases, the matched node will be the root node or a leaf of the tree, so there are two ways to expand this tree, as exemplified in figure 9-3. If the matched node was the root node, the newly connected node will become the root node and the matched node its child; else the node will be added as a child to a leaf. If the node is connected to a node that already has children, it will be added as one of the children and the tree will branch. After adding the node, this connection is removed from the results and the algorithm will look at the next result.

Once the complete set of results has been walked through, but there are still a number of connections left, the algorithm will compare the number of results before and after the walk through. If the number is different, then the tree will have been adjusted during this walk and there will be new possibilities for connections to be made. The algorithm will then walk over the results again in the same way. However, if the number has not changed, no further additions can be made, as the state of the tree has not changed. Since there are still connections left that cannot be added to this tree, a new tree will be started to be able to interconnect the rest of the nodes. This process is repeated until the list of connections is empty and we have one or more trees ready.

These trees, however, do not correspond directly to the division into segments, into which paths are divided. This division is made explicitly by another algorithm that starts at the top node and creates segments for each combination of top and bottom node it recursively comes across. The children of the bottom node of the current segment will be the top nodes of the child segments. So the spider cable in figure 9-6 would transform to two segments that both have the first segment as a parent and both have a segment as their own child.
Figure 9-5 The conversion of a node tree to a segment tree

Figure 9-6 The conversion of a branched node tree to a branched segment tree
SQL for selecting segments

-- In the selection block all the information that is needed for the algorithm is selected. See also the records described in the text.

SELECT DISTINCT
N1.nodeId, N2.nodeId,
C1.connectorId, C1.twelveNc, C1.xnr, C1.location, C1.type, C1.module,
C1.gender,
PN1.pinNumber, PN3.pinNumber, PN3.signalName,
C2.connectorId, C2.twelveNc, C2.xnr, C2.location, C2.type, C2.module,
C2.gender,
PN2.PinNumber, PN4.pinNumber, PN4.signalName,
C1.cableId, CB1.twelvenc, CB1.sapName, CB1.sacaId, CB1.length,
CB1.kind, CB1.module, CB1.multipleVersions,
C2.cableId, CB2.twelvenc, CB2.sapName, CB2.sacaId, CB2.length,
CB2.kind, CB2.module, CB2.multipleVersions

-- The algorithm will need the connections, between two connectors that each have a cable and a pair of pins

FROM
ConnectorToConnector CTC, Connector C1, Connector C2,
Node N1, Node N2, Cable CB1, Cable CB2,
PinNode PN1, PinNode PN2, PinNode PN3, PinNode PN4

-- The condition starts with a connection, of which we need the connectors, their nodes, their cables, and their pins and return pins

WHERE
CTC.connectorId1 = N1.connectorId
AND  CTC.connectorId2= N2.connectorId
AND  N1.connectorId = C1.connectorId
AND  N2.connectorId = C2.connectorId
AND  C1.cableId = CB1.cableId
AND  C2.cableId = CB2.cableId
AND  N1.nodeId = PN1.nodeId
AND  N2.nodeId = PN2.nodeId
AND  N1.nodeId = PN3.nodeId
AND  N2.nodeId = PN4.nodeId
AND  PN1.signalName = 'signalName'
AND  PN2.signalname = ' signalName ')
AND  NOT( PN3.signalName = ' signalName ')
AND  NOT( PN4.signalname = ' signalName ')
Algorithm for filling the path
results = get the path segments from the database;
while (there are results left) {
    for (each result) {

        //Each result is a tuple of two nodes on two connected
        //connectors(node1, node2).
        get the fields from the result
        construct the cables and nodes

        //If there is no tree yet, a new tree should be started from the
        //current nodes.
        if (it is the start of a new path) {
            Start a new tree;
            Make one node the root of the tree;
            Add the other node as the child to the tree;
            remove this pair from the results;
        }

        //If there is already a, tree this part tries to find a node
        //matching one of the result nodes and add the other as a child.
        //If they are both in the tree already, we connect them.
        else {
            if (node1 is not the tree) {
                if (node2 is in the tree){
                    match = node2
                    nodeToAdd = node1;
                }
            }else if (node1 and node2 are in the tree){
                connect node1 and node2;
                remove them from the results;
            }else{//Only node1 in the tree
                match = node1
                nodeToAdd = node2;
            }

            //If a match is found, we add it on the top (before the root)
            //or bottom (after a leaf).
            if (a match was found){
                if (match equals the root) {
                    set nodeToAdd as new root and the old root its child
                }else{
                    add nodeToAdd as the child of the match
                }
                remove this pair from the results;
            }
        }

    } //end for

    if (no matches were found or trees were made for all results) {
        the next result will be the start of a new path;
    }
} //end while
Algorithm for creating the segment tree
Node topNode = root of a tree of connected nodes;

// If there are no children, a segment without bottom node is created.
if (topNode has no Children) {
    return a segment with only a top node and a “no end node” exception
}

// visit the connected nodes of the topNode to make as much segments as there are branches.
for (each bottom node) {
    Create a separate segment with the topNode and bottom node
    for (every child of a bottom node) {
        create the segment tree for this top node;
        connect this segment tree to the bottom node
    }
    if (there is more than one bottom node) {
        add spider cable exceptions to the bottom node
    }
}
return the list of topNode-bottom node segments;
10 Application

10.1 Introduction
The application was originally meant as just a demo-tool, but has gained more functionality along the way. It will mainly be used to show what the possible uses of RIP are, but has turned out to be a tool that can be improved to ASML standard software. This chapter contains the five different screens that this application has and their functionality and, if applicable, the algorithms that create the visuals.

10.2 Login screen

**Description**
The login screen pictured in figure 10-1 allows the service engineer to gain access to the application. Users have to provide a username and password to log in. This procedure is only a basic security measure, to prevent unlimited access to the functionality of the tool. When logging in, the user will also need to select the machine that they want to use. This machine field is not used in the application at this moment, but will be in the future. The password will be hashed with SHA1 and sent to the authorization algorithm that will set the general user type for all the other screens being: Administrator, Moderator, Extended user or Service Engineer. After logging in, the screen will show the username and a logout button as in figure 10-2. Once the user has pressed this button and it logged out, the user type will be cleared and all rights will be removed.
10.3 Importing

Description
After the user has logged in, they will have access to the import screen. The import screen has one import button and four buttons for exception checking, which are enabled or disabled based on the type of user. Each check for an exception provides a log file with the name of the exception and the date in the following format: YYYY-MM-DD_hh-mm-ss_Exception.txt. This allows them to be sorted on the date to easily retrieve a log. They are stored in the same directory as the program is stored in.

The log files are preferred to database entries, because this eliminates the need for the tool to be available to read a log. This way it is easier for service engineers to find, read, and copy logs to others. In a later stage, the logs could always be exported to the database by simply defining a function in the LogFile object and creating a table in the database.

The console window is for showing messages regarding the status of the import process and the name of the created log file afterwards. It catches the System.out stream of Java, so the system does not have to start a separate logging thread for only this purpose.

Rights
Only Moderators and Administrators can import the data into the database and check for exceptions. Service Engineers and Extended Users are not allowed to make any changes to the database.
10.4 Configuration

**Description**
In the configuration, it is possible to set the paths to the directories that are used during the import, and the path to the directory for the website. An administrator can add extra users.

**Rights**
Only the administrator can use the configuration screen to add or change values in the database. Moderators should only be able to access the information from data sources, i.e. importing data and checking for exceptions and later on make manual changes to the database via the tool. Service Engineers and Extended Users should not be allowed to make changes to the database in any way.

![Figure 10-5 The enabled configuration screen for Administrators and Moderators](image)

![Figure 10-6 The disabled configuration screen](image)
10.5 Searching

Description
A user can search for signals or other objects by selecting the option ‘signal’ or ‘other’ in the combobox in figure 10-8. Searching through the data itself happens as described in chapter 9.2. The results of a search will be shown in a clickable list in the grey box, as shown in figure 10-7. When a result is selected, by clicking once, the properties will be shown in the greenish box, as in figure 10-9 and figure 10-10. Only if the result is a signal, no extra information about a selection will be shown. Consider: Double clicking on a result will only be possible when it is a signal. This action will open the map that belongs to this signal.

Rights
Everybody can do a search, but only the Extended User and users of a higher rank can view the links to the outside sources in the properties.

Figure 10-10 The empty search screen

Figure 10-10 The search for a signal

Figure 10-10 A search for cables and connectors. A cable is selected.

Figure 10-10 A search for cables and connectors. A connector is selected.
10.6 The map

Description
Figure 10-11 pictures an overview of a map screen that is shown after double clicking on a signal. The map screen shows the list of segments that belong to a signal, together with the lengths and properties per segment. Each part of this screen is explained further in the following chapters.

Figure 10-11 The overview of the map screen
Header

Adapter Offset: [Field to enter adapter cable length]

Length:

Signal: ISWSTISA
Return: ISWSTISB

Additional Info

On the left side of the header there is a text field in which the user can put the length of an adapter cable from the tool to the path that is to be measured. On clicking ‘GO’, RIP will update the lengths in the left-side column.

In the centre the signal and return signal are displayed, with return signals separated by commas if there would be more than one.

Elements

Pins

The pins are shaped like pins on the wiring diagrams, so the service engineers, will recognise them soon. The pin with the curve on it, is the female pin, the other the male pin.

Node (Connector)

The node is represented as a connector from the wiring diagrams, so it mostly looks like a connector with two pins.

Segment

The segments on the wiring diagrams are just lines, but using a filled square with information to connect two nodes is clearer than a line with information next to it. It has the Xnr on the top and bottom and the SAP name in the middle.

Properties

The properties in the light grey area belong to respectively the top and bottom node of a segment. The dark grey part belongs to the cable in between. All middle segments connect to each other, so the top and bottom node of two connected segments will form a field with the four properties that belong to this connection.

If a segment is the very first or very last of a path, the properties of either the top node (first segment) or bottom node (last segment) will be extended to contain all the information about this connection. Even though the node seems disconnected in the map, in the machine it is connected to the other gender version of itself on its location.
Length

The length is given in panels alongside the segments. Each panel contains the length of the segment itself, the length of the path before the segment in the right row and the length of the path after the segment in the left row. Both directions are needed, because it is unknown from which side of the path the service engineer starts its measurement. Note that in this picture the example is cut out from the overview and does not represent the complete length. The left row starts at 0.0 m and not at 1.3 m. The right row will eventually end at 7.73 m.

When we take the first segment as starting segment for this example, the right row gives us 0.0 m as distance to this segment and 6.43 m as distance after this segment is traversed on the path. The path will then still have 1.3 m left to traverse. The left row gives us distance 1.3 as distance to this segment, when we would start from the bottom segment and 7.73 as distance after this segment is traversed.

Visualising the segment tree

The biggest part of the creation work is done in the Controller, to create the tree of visual representations of segments (in SegmentView) with the data from the model. The Controller walks through the tree of segments and converts each segment directly to a SegmentView, the nodes to two attached NodeViews and the pins to attached PinViews. This way a tree of connected SegmentViews is constructed, but these have only the data from the model.

The positioning of the SegmentViews is done separately using the setSegmentCoordinates function. It recursively walks through the tree and sets the coordinates of each SegmentView on basis of the coordinates of the parent. The room this tree of SegmentViews has in the width is called the ‘lane’. The whole of the tree thus cannot exceed this lane and will be centred within it. If the function notices a split in the tree it divides the lane into two lanes and centres each sub-tree within its own lane, which is thus half the space the lane of the parent had. The algorithm will save the maximum depth to make the scrolling pane on which the tree is drawn go deep enough. In figure 10-12 a tree is pictured in which the lane has been split twice: once after the second node and once after the first node of the left sub-tree.

In the last step in the controller, the view is changed from the search view to the map and calls the repaint function of the map, so all parts will be drawn by their draw functions. In this whole process no new data may be added, as all the data should be available from the model.

Rights

Everybody can view the map, but only the Extended User and higher ranked users can view the links to the outside sources in the properties.
**Algorithm for creating the path**

{  
    Start a new pathView on the map  
    myPath = a new Path for this signal;  
    for ( each segment tree ){  
        make a SegmentViewTree with the root of the segment tree;  
        add the SegmentViewTree to the pathView;  
    }  
    Change the view to the map view and redraw;  
}

**Algorithm for creating the SegmentView tree**

Converts the segments, nodes and pins, to SegmentViews, NodeViews and PinViews in a similar tree structure.

{  
    create a new SegmentView;  
    for ( each segment node ){  
        Create a nodeView of the node;  
        for ( all pins from the node ){  
            Add a new pinView to the node;  
        }  
        if ( node is topNode )  
            make it a topNode in the segment;  
        else  
            make it a bottom node;  
    }  
    for ( each segment child ){  
        make a SegmentViewTree of the child;  
        add this tree to the segmentView  
    }  
    return the segmentView;  
}
Algorithm for drawing the pathView

In this algorithm first all the coordinates of the segments will be set to be able to draw them separate from their tree structure. Before the first tree a header is added to the map, after that for each segmentView a panel is made for the segment and the properties. In the end the cumulative lengths of each segment are converted to panels. The lengths and properties are only added for the largest branch of the tree if there are subtrees, to conform to the user interface of the prototype. If there is more than one path, a broken path panel is added in between.

```java
{ Start a pathPanel
  for(all trees of segmentViews) {
    Set the coordinates of the segmentView tree;
    Start the array of cumulative lengths with 0;

    for(all segmentViews of the tree) {
      if(it is the first segmentview of first tree){
        Create the header panel and add it to the pathPanel;
      }

      Create and add the panel of the segmentView;
      if(not on the same depth as an already visited segment) {
        add this length to the cumulative lengths;

        //create a different propertiesView at the top and end
        if(first segmentview of this tree){
          Create top properties panel
        }else if (last segmentview of tree){
          Create bottom properties panel
        }else{
          Create middle properties panel
        }

        add the properties panel
      }
    }
  }

  Create and add lengthpanels from the cumulative lengths
  if(this is not the first tree){
    create and add a broken path panel
  }

  return the pathPanel;
}
```
Algorithm for setting the coordinates of the SegmentView tree

This algorithm starts at the root of the tree and takes a segmentView, an x-coordinate that indicates the offset from the left, a y-coordinate to indicate the offset from the top and a laneSize variable that indicates the size of the lane this tree is in.

```
{ segmentView.setX( ( x + laneSize - size of segment ) div 2); segmentView.setY( y );
  if (the segment has children){
    laneSize = laneSize divided by the number of children;
    for (int i=0; i < number of children; i++){
      set the segmentViewTree coordinates for child[i] where:
      the x offset = i*laneSize
      the y offset = y + segment height
      laneSize = the newly calculated laneSize;
    }
  }
}
```
11 Conclusion

11.1 Summary

The initial aim was to find out if it was possible to develop a program that automatically imports routing information from several sources to construct netlists that can be used by the service engineers. The sources are prone to errors and some known errors have to be detected and handled by the program.

A study on deductive databases showed that there are several usable databases that run on Prolog and can be used for this purpose. Possible databases were IRIS and SWI-Prolog. In the end, however, Oracle in combination with JAVA was preferred, because it already belonged to ASML's set of companywide available software.

The developed application consists of a 3-tier architecture with the layers: Database, Data and application logic, and Presentation. The Model-View-Controller pattern is used to decouple the data and the view and the decomposition in packages is done according to this pattern.

The import functionality can import several data sources into the database. This project has proved that it is possible to extract all needed data, but the extraction of some data (especially lengths from EDArch files) still has to be implemented.

To be able to handle the error prone files, a set of integrity checks is implemented that detect and correct, highlight or explicitly ignore the following exceptions:

- Spider cables
- A signal changes its name within a location
- There are multiple pins for one signal on a connector
- Mismatched genders of pairs
- A general breach in the path of a signal, for example by
  - Discontinuity between Cos
  - An unforeseen change in Xnr or 12NC in one of two connected connectors.
- Cables that have only connector, instead of two or more.
- Cables that have multiple versions

The functionality that the user will use most is implemented in a standard searching facility in which the user can search for 12NCs, SAP names and signals. It returns properties of cables and connectors and for a signal the map of the signal can be generated. This map consists of a path of segments the signal runs over with the cumulative length and the component's properties next to them.

This all forms an application that takes care of the whole automation from getting from the data sources to a usable map of a signal that can be used in combination with a TDR tool to find a breach in seconds and reduce repair times drastically.
11.2 Future Work

The work on RIP is far from finished. This is the first step to prove the feasibility of the project and to raise the awareness about the issues this tool can solve, which raised a lot of enthusiasm among other divisions of ASML that now see the potential time and cost savings. Apart from the tutors (Of which Rene has taken the role of client and Johannes the role of enthusiast pioneer) also people of other groups now would be very grateful to have this tool to test their data. Among them are:

- Leonie Jansen of CSI project management (Cross-sector Structural Improvement)
- Vincent Kok of CSI Engineering

With among others those people and the two tutors of ASML, a list of new requirements for the future will be made, but a list of some important issues that still require further investigation and/or implantation is already available:

Importing

The importing of EDArch files is already described and partly implemented. Connecting the EDArch signals to the cables only needs implementation and the gathering of lengths from the files still needs investigation and implementation. TCE and CO’s can already be imported, but some investigation on version control is still needed to let all the different CO’s work together seamlessly.

Database

The database itself still needs a good implementation of version control to be able to save changes to the database instead of just one backup as described in chapter 7. These changes can also be manual changes that can be done at a new editing screen in the application. Version management and incorporating manual changes could also both be moved to a separate maintenance tool for only the administrators, too keep the application itself lean. It would furthermore contribute greatly to the simplicity if the program could use one database per machine instead of one database for all machines.

The database probably has to be replaced by the free database MySQL, for laptops without an internet connection. Once the queries are converted to SQL syntax, the program will work the same as before. MySQL is the standard free database within ASML and is used for having a local database on a computer. Other options for local databases are free database systems that are designed for working in a local environment like SQLite.

Maturity

The tool has to be modified to fit into the ASML tooling range. This means that the interface has to be standardized, but also that the authorization code has to be updated to make sure no one can directly access the database after they have this program delivered on their disk. Especially a local database should be secure enough to send into the field.
A. Task analysis

This is a summary of the task analysis that was done to find out how a service engineer can profit from this tool when a part of the machine fails.

Case

This program has many uses, but its purpose is derived from the case that a part of the machine, say a sensor, fails and that the only info available is the 12NC of the cable connected to that part.

Questions to which the service engineer needs an answer

• What is this sensor used for?

• Which sensor board is used to read the sensor values? I.e. where is the end point of the cable between the sensor and the component that uses this sensor.

• How is the cable routed through the machine?

Tasks the service engineer will perform in the search of this answer:

• Find the routing of the cable until an electronic board or iPCB.

• The board will be looked up by 12NC, so the engineer can look up the other cables that are connected to this board and find the right cable to follow based on the signal name. Sometimes the signal will form a pair with a return signal and sometimes it will be just an individual signal.

• If a cable is found (by 12NC), all signals of the cable will be looked up and the engineer will create a complete netlist by following the signals through the wiring diagrams.

• A service engineer might also want to try and find the netlist of a parallel signal to compare it against the current netlist.

• From the netlist, the engineer will follow links to other documentation for more information about a specific component.

• These diagrams will almost certainly contain errors that the service engineer might need to solve by reasoning, but on which an automated program would break.
B. User requirements document

This is a summary of a 24 page document that was made to contain the wishes of the client and the general description of the problem the client is hoping to solve, the use cases that explain how the program will be used and the environment it will work in. It is based on the ESA standard for an URD found at http://wwwis.win.tue.nl/2R690/urd.html.

General description
The main goal of the program, the four users and the environment in which RIP will do its work are described here.

Use cases
Use cases describe the actions that a user could perform through the user interface.

Functional requirements
The actual requirements that describe what the system has to be able to do. They are divided in requirements about the database, importing, searching, exceptions, path building objects (signals, segments, nodes, connectors, cables, components and paths), the general user interface and the map. Moreover, some loose requirements that are already stated, but will not be implemented are described as ‘could-haves’.

Non-functional requirements
The non-functional requirements are to describe the software language, documentation language and Operating system it should work on. No specific performance constraints have been given in this section.
C. Software requirements document

This is a summary of a 51 page document that was made to contain the wishes of the client translated to classes and functions and the scenarios of how it will work translated to the interactions the software has within and with the user. It also contains a prototype of the proposed application. It is based on the ESA standard for an SRD found at http://wwwis.win.tue.nl/2R690/srd.html.

**General Description**
The general description describes the relation to other projects and systems. There is no relation to other projects within ASML, but there are relations to 4 other systems: cable overviews, TCE, SAP and EDArch. It also describes the function and purpose of the program, consisting of a database, API and demo tool as proof of concept.

The model description describes the use cases in terms of software calls and messages, using message sequence charts. The database model describes the dependencies, cardinalities and keys as shown in chapter 7. The class model describes the dependencies between the classes and the functionality of each class. The dependencies between the classes without the functionality are visible in chapter 5.2.

**Specific requirements**
The specific requirements convert the user requirements each to one or more functions or variables in the classes. Of each class the purpose of each function is described. The complete functionality of all the classes as implemented now is appended as Javadoc.

**Prototype**
The prototype consists of screenshots that tell what each user action would cause the system to do and what that would like on the screen. The prototype also contained a framework of the program that (besides showing a map) imitated the functionality that should be implemented and that has been improved to a working demo tool.
D. Architecture design document

This is a summary of a 20 page document that was made to contain the architecture of the software framework, which methods and patterns were used for the design, how it will work together with existing systems and the resources it needs to work properly. It is based on the ESA standard for an ADD found at http://wwwis.win.tue.nl/2R690/add.html

System context

The system context describes the other systems within ASML that this systems will work with. In this case the cable overviews, SAP, TCE, EDArch and the Oracle database.

Design method (copied from ADD)

The system is designed in an Object Oriented way and represented by the various UML diagrams. The Model-View-Controller design pattern is used for the decomposition of the components to minimize the coupling of the model from the application, as the application will merely be a demo-tool that can be build out, but just as easily replaced by a new application.

Decomposition description

The decomposition description describes the decomposition in packages of RIP and the interfaces that each package offers to the other packages.

Feasibility and resource estimates

Here the resources are estimated that are needed to develop and to run RIP
E. Definitions

12NC – The 12NC is a 12-digit number that uniquely identifies a part, such as a cable or interconnection circuit board. It is formatted as xxxx.xxx.xxxxx with x being a digit, like 4022.567.89101. The first 4 digits always form the prefix “4022”, which is the standard prefix for all parts within ASML’s machines. The next 7 digits identify the part and the last digit identifies the version of the part. The version number always starts at 1, so the example 4022.567.89101 would be the first version of this part.

API - An Application Programming Interface is a particular set of rules and specifications that software programs can follow to communicate with each other.

Cable - A cable consists of one segment that connects one or more pins/holes in a connector on one side to one or more pins/holes in a connector in the other side. Each cable can be identified by its 12NC. Also see Overview of the machine.

Cable overviews (CO) - A CO is an overview in Visio of all the cables that connect the components to each other. Also see Overview of the machine

Component - A part in the machine, like a cable, sensor, PCB or assembly. Each component can have a 12nc, which uniquely identifies the component. Each component has one or more connectors that can be connected to the start or the end of a path.

Connector - A plug with a number of pins or little holes, found at the end of a cable or connected to a node. Each connector has a unique combination of component it resides on, Xnr and gender. Also see Overview of the machine

Datalog/Prolog - Datalog is a query and rule language that is based on Prolog which in turn is based on first order logic. It is used in deductive databases: databases that can derive new facts out of existing facts in combination with rules on those facts. It is especially suited for recursive queries of which you do not know the length beforehand, like calculating all the ancestors of a certain family member or all the paths between two points.

EDArch – An archive that contains all Electric design files. Contains among others files with the pin-signal combinations and lengths of the circuit boards.

Gender - A connector can have a gender: male means the connector has pins, female means it has holes.

iPCB - Interconnected Printed Circuit Board. An iPCB is used to connect two or more cables or iPCBs. It has connectors on both sides and a board with interconnections in between. It should be treated as an (advanced) cable.

Location - The part that a connector is connected to. The location can be any kind of component that interconnects two segments.

Netlist - A list of segments and nodes that are connected to each other. The netlist contains the routing information, which should contain the nodes, segments, signal and length information,
and a good indication of where to find them in the machine. In the application it is represented in visual form as the map.

**Node** - A node consists of a number of pins on the connector of a cable or interconnecting component (usually a pair of pins). Segments are connected to each other by the nodes on their ends, like this: segment–node – node–segment.

**ODBC** - Open Database Connectivity is a standard software interface for accessing database management systems. The designers of ODBC aimed to make it independent of programming languages, database systems, and operating systems. Thus, any application can use ODBC to query data from a database, regardless of the platform it is on or DBMS it uses.

**Parallel path** - Two paths are parallel for the part that they run between the same connectors and thus follow the same route.

**PCB** - Printed Circuit Board. A board with circuits on it.

**Pin** - A little piece of metal sticking out from a connector that can be fitted into the receptacle of another connector. Commonly connected to one line of a cable and forming a pair with another pin in the same connector.

**RIP** – Routing Information presentation. The name for the whole of this project.

**SACA-id** - An identifier of the cable. If a cable has a SACA-id it means that the cable is a standard cable and uses 1-on-1 pinning. Pin 1 on the connector on one side, should always transport the same signal as pin 1 on the connector on the other side, and so on for every pin.

**Signal** - A named electric signal that is commonly sent from a starting point over a path of segments to an endpoint. Commonly belongs to another signal that goes from the endpoint to the starting point. When signals belong together they usually have similar names like SIGNALNAMEp and SIGNALNAMEn to denote that there is a positive and negative signal that belong together.

**Segment** - A segment is line that is a part of a cable and usually connects two pins from one connector to two pins of another connector, though exceptions are possible. A couple of segments together provide a path for a signal.

**Sensor** - A device that sends electric signals based on changes in its environment. Has connectors to connect to cables.

**STRIP** - The name of the Oracle database behind the application.

**TPD** - Technical Product Documentation. All technical documentation that is connected to a product.

**Xnr** – The number to identify a connector on a part. Usually starts with an X like ‘X10’, but this is not mandatory. Each Xnr-gender combination is unique on a single part, but it can be used again on another part.