"Großer Beleg" Thesis

Cost Analysis for Computer Networks with XML-DB and Java-Applications - Implementation in the frame of pilot project CANDY

Ulrich Winkler

Supervisors: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Dr.-Ing. Andriy Luntovskyy
Dr. re. nat. Dietbert Gütter

Issued: 01. 06. 2005
Handed in: 15. 01. 2006
Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

I declare to have written this work independently and without using unmentioned sources.


[Ulrich Winkler]
Acknowledgements

I would like to thank Prof. Dr. Schill for the opportunity to work on this project. My special thanks go to my supervisors Dr. Luntovskyy and Dr. Gütter who always had time for a discussion. I would also like to thanks my sister Gudrun "Runi" and her fiance Marko "Ruscmuba" for helping me to correct my poor English.
## Contents

1 Introduction ........................................... 1

2 Related Work ........................................ 2

3 Design .................................................. 11
   3.1 CANDY’s architecture nowadays .................. 11
   3.2 Components related issues ......................... 13
      3.2.1 Editors for a cross document model .......... 13
      3.2.2 RuleChecker ................................... 15
   3.3 Cost accounting related aspects .................. 16
   3.4 Conclusion .......................................... 18
   3.5 Candy Platform ...................................... 19
      3.5.1 Eclipse RCP ................................... 20
      3.5.2 EMF - Eclipse Modeling Framework .......... 23
      3.5.3 SimpleCore and Net Editor ................... 24
   3.6 Cost editor and the cost model .................... 25
      3.6.1 BIRT - Business Intelligence and Reporting Tools 26
      3.6.2 HSQLDB - Hypersonic SQL Database .......... 29

4 Implementation ....................................... 30
   4.1 Resource Manager and the SimpleCore model .... 30
   4.2 The Database Adapter .............................. 39

5 Conclusion ........................................... 45
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The graphical model as designing in [Sch02]</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>The <em>workplace</em> as represented in [Sch02]</td>
<td>3</td>
</tr>
<tr>
<td>2.3</td>
<td>Client-Server architecture as shown in [Sch03]</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Layered architecture as described in [Grü04]</td>
<td>6</td>
</tr>
<tr>
<td>2.5</td>
<td>The <em>viewpoint</em> concept</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>Candy Version 2 architecture as developed in [Zho05]</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>CANDY Platform design overview</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>CANDY Platform design details.</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>To create assertoric business reports BIRT uses the NDM and other data sources, like databases and log files.</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>The NDM adaption through a Database Adapter (DBA).</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>The NetEditor and the SimpleCore editor</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>The layout view of the BIRT editor.</td>
<td>48</td>
</tr>
<tr>
<td>5.3</td>
<td>The preview view of the BIRT editor.</td>
<td>49</td>
</tr>
</tbody>
</table>
1 Introduction

Since 2003 the project exists CANDY at the Dresden University of Technology. Aim of this project is to provide an integrated design environment for designing computer networks. Foundation for this environment is a XML based model language and a network model.

One issue is the financial aspect not only in the design or deploying stage of a network’s life cycle. Financial and budgetary controllers demand a detailed and adapted real-time cost reporting- and billing system. This work shows how to design, create and deploy business intelligence reports in such cases as real-time demands.

Since the very beginning every new implemented part of CANDY used another technology, XML description or design to deploy additional functions. One part of work within this thesis was to describe a platform which should lead to a convergence process in the development of CANDY.

A short overview of nowadays CANDY is given in chapter 2 Related Work. In chapter 3 Design the platform is described and the striven integration of a cost and billing tool is outlined. Furthermore in chapter 4 Implementation some details are given about the implementation of the platform and the cost tool and in chapter 5 Conclusion the obtained results are outlined and discussed.
2 Related Work

This chapter gives a short overview over the development history of CANDY with attention to architecture, model and support of a billing tool.

Peter Schmid, "Projektierung von Unternehmensnetzwerken und Konzipierung einer Entwicklungsumgebung" [Sch02] introduced a model for designing computer networks according to EN-50173. This model was named the "graphical model". Every object has its own unique identifier - abbreviated "ID". Figure 2.6 shows a diagram of the proposed graphical model. The ID should be used to receive additional attributes such as price or length of cable. Every object should be stored in a database with its unique identifier as reference.

A graphical editor and a persistent form for the graphical model were introduced. The graphical editor should be the main editing part of the application and mainly arrange the nodes and arcs of the graphical model, and help the user assign attributes. The persistent form (or the serialized form) of the graphical model is saved into a XML file¹.

Further applications, such as RuleChecker, BillRePoter and others, use this serialized graphical model and produce additional documents. In [Sch02] it was not mentioned if those applications extend the graphical model with their specific information or not.

¹ [Sch02] used Microsoft Visio as the graphical editor and therefore there is a need for an additional tool, named Extractor in order to get rid of the unneeded XML tags in a Microsoft’s Visio produced VDX file. This Extractor produces a perfect XML file. This has the disadvantage, that you never can change a "perfect XML file" and read the changed "perfect XML" file back in its original editor - in this case Microsoft Visio.
2 Related Work

**Figure 2.1:** The graphical model as defined in [Sch02]. The yellow boxes are nodes, the turquoise are links and the big red box is a connecting devices. Note, that there exists no direct path device - cable - device and therefore it is not possible to model a bus cabling computer networks. It is a right, because [Sch02] relates only to the EN 5017 standard.

**Figure 2.2:** The workplace as represented in [Sch02]. The workplace and the graphical editor running local on the users machine and using a local accessible database. The workplace consists of several modules, like the BillReporter or RuleChecker and Document Generator.
All application should be integrated within a program named *workplace*. This workplace is responsible for the coordination between the user, the database, the serialized graphical model, the graphical editor and all other parts belonging to this product. Figure 2.2 shows the relationship between all components.

The task of the BillReporter is according to [Sch02] to map all components contained in the serialized graphical Model onto a price value and summarize it. For cables the user must assign the additional cost value, a price per meter. The approach is convenient because it is uncomplicated. The disadvantage is, that it is not flexible enough to cover all real-life requirements.

**Gert Grunewald, "Performance simulation mit NS-2"** [Gru03] showed how to transform a XML based network description via XSLT into a NS2 simulator readable OTcl-script file, where a simulation is run and interprets the generated resulting trace files. There are several points of interest for a billing tool:

1. Several estimated traffic network values, e.g. outgoing traffic over a link where you have to pay per volume

2. Estimated traffic in different networks, e.g. the traffic in all LANs, the traffic between LANs and the traffic into maybe different WAN. This differentiation is necessary, because some providers offer ADSL products, where you have different prices for each volume unit. For instance some ISP charge different prices for traffic into the provider network, the country network, or foreign networks. This is common for example in Portugal or the Ukraine and differs from most German providers, where it does not matter if the traffic is within the provider’s network or not.

Some traffic limits, like: "the first 1GB is free, then you have to pay 1 Euro Cent per Megabyte" may also be taken into account.

---

2 Today the Portuguese provider SAPO offers a ADSL product: up-link 2048 Kbps, down-link 128 Kbps, monthly cost 35,58 EUR (includes 20 GB national traffic and 2GB international traffic; additional costs are taken into account if you exceed those limits: 1.5 EUR per 100 MByte international traffic and 0.10 EUR / 100MByte national traffic)
Figure 2.3: The application design differs in [Sch03] from [Sch02]. Now CANDY run on an application server. The graphical interface sends all requests to the server. A coordinator collects the request and delegate them to the most eligible module. The result is represented the user in the graphical interface.

Peter Schmidt, "Entwicklung einer XML-basierten Arbeitsumgebung für die Rechnernetzprojektierung" CANDY was mentioned for the first time in [Sch03], wherein the given definition of CANDY is not very precise at all. The design approach of CANDY was also changed towards a web based application. The client is a java applet and the whole application is running on an application server ([Sch03] proposes Jakarta Tomcat). As you can see in Figure 2.3 the graphical interface does not execute any business logic.
Likewise the graphical model as described in [Sch02], was replaced by an *abstract network model* and a concrete Java implementation. The model is saved by serialization of model class into different XML files. For each XML file a DTD is given.

There is no build-in support for billing analysis, but in [Sch03] the process of localization and translation is explained. For a billing tool localization it is important in order to present values with different fractions, currency units and even different icons for currency units on buttons. The proposal made in [Sch03] is simple, but not sufficient enough. [KFL03] shows various aspects of internationalization and localization. A tutorial is available at http://java.sun.com.

*Figure 2.4:* layered architecture as described in [Grü04]
CANDY: Integrated Environment for network design

In [LGP04] the concept NDML and its concept of viewpoints are introduced:

NDML "Network Design Markup Language - is a XML based problem oriented design language, hierarchical and extensible, able to describe basic standards and organizational areas (campus, building, floor), structured cabling systems, data transmission media among other things for mobile and wireless links, active network coupling devices, network nodes and their functional dependencies and parameters, heuristics about customer-specific user applications and their caused network traffic used for simulation, data export/import interfaces and protocols for compatibility software as well as to encapsulate remote calls of design applications."

NDM "The informational model (NDM - Network Design Model) is built on the basis of the topology description of the network augmented with functionality of each concrete node, and its parameter."

Viewpoint A Viewpoint is a XML file, descriptive a special aspect of the NDM and is part of the set of other Viewpoint files. This set is also known as NDML. In [LGP04] seven viewpoints are defined:

1. Basic - basic project descriptions
2. Geometry - rooms, buildings, floors, etc
3. Topology - network technology, network interfaces, etc
4. Cost - cost of devices, cables, etc
5. Load - bandwidth, burst times, traffic type, etc
6. Queuing - load, delays, etc
7. Simulation - protocol and application agents, etc

Figure 2.5 shows the Viewpoints.

[LGP04] also mentioned the use of XSL to transform parts of the NDML into different programming languages like Java or C++ in order to create runnable units.
2 Related Work

Figure 2.5: The viewpoint concept. According to \cite{LGP+04} a viewpoint is a design stage. During a design process a user enter a viewpoint, transform the NDM, and leave the viewpoint. Viewpoints have dependencies among each other.

and to gain simulation results. \cite{Gru03} and \cite{Fäl03} used this approach to create simulators.

Wolfgang Fäller, "Analytische Verfahren für die Kapazitätsplanung von Rechnernetzen auf Basis der Bedienungstheorie" \cite{Fäl03} showed a solution similar to \cite{Gru03}. The difference is that \cite{Fäl03} produced a C++ source code file out of a NDML file This C++ source code has to be compiled and linked against a library for which purpose it is developed for. The execution run of the software produce also a trace file like in \cite{Gru03}.

Artem Kaftanenko, "Methoden und Algorithmen zur Optimierung der Leitungsleitung in lokalen Netzen" \cite{Kaf04} actualized a prototype to handle the complex topic of modeling cable plumbing in a three dimensional environment like a building. \cite{Kaf04} introduced a building model and different network model concentrating on technical devices like patch fields and plugs.

A graphical editor from the scratch was also designed for editing this model. \cite{Kaf04} mentioned that this editor may be extended by other modules.
This model provides several points of interests for a billing tool, e.g. cable ducts, patch fields, sockets, the real length of a cable in-wall, etc.

**Feiyue Zhou, "Entwicklung einer XML-basierten Arbeitsumgebung für die Rechnernetzprojektierung"**  Aim of [Zho04] was to design and implement an integrated environment for CANDY tools. Similar to [LGP+04] this was achieved due to a web application.

A network editor is also provided. [Zho04] used the model designed by [Sch03].

This work seems to be a good starting point to integrated a billing tool into the CANDY framework.

**Heinrich Grünewald, "Kostenanalyse bei RN - Projektierung auf Basis von NDML"**  [Grü04] wanted to contribute the viewpoint 'cost' as defined in [LGP+04]. Therefore, [Grü04] separated five different aspects which have to be taken into consideration:

- cost of planning
- cost of installation
- operational cost, cost of after-sales service, depreciation cost
- investments, renewal or maintenance costs

[Grü04] also extended NDML for the cost viewpoint. A attribute *costs* was added to the elements *device*, *medium* and *nic* A servlet was also given, which allows to upload a NDML file, to add mappings between devices, mediums, NIC and an entry in a RDBMS. Hence one can assign a product description to a device with its price. The resulting costs are added to the related tags in the NDML file. The total costs are represented on a website, where the transformed NDML file can be downloaded. A Microsoft Access 2000 database is used to store product information.

**Torsten Haase, "Dimensionierung drahtloser Campusnetze auf Basis von Hotspots"**  [Haa05] explained wireless LAN technologies and how this can be modeled with CANDY. An extension to the already existing NDML is proposed, called
"RadioNDML", and combined it with a model by [Kaf04] in order to describe buildings. Prospects for this extension were to have a handy model to search optimal locations in order to install wireless access points. Furthermore *Ekahau survey* is supported, a comprehensive tool for 802.11a/b/g network site surveys and optimization of WiFi networks.

**Thomas Fahnert, "Analytische Verfahren zur Ermittlung optimaler Standorte von Hotspots"** [Fah05] programmed a tool to find a perfect spot for wireless access points. He used PhytonCad as foundation.

**Feiyue Zhou, "Entwicklung und Optimierung eines graphischen Entwurfssystems für das Rechnernetzprojektierungstool CANDY"** The architecture of CANDY was changed by Feiyue in [Zho05]. The most important changes to [Zho04] are that lot of business logic and presentation is transferred back to the client and furthermore, that applets are not used anymore. The net editor, for instance, is a java software application. The new clients take advantage of CANDY Manager, a server side resource and user management system.
3 Design

Within this chapter problems related to the design of CANDY are highlighted. An initial stage towards a solution of the disclosed problems is given and a primary integration of a billing tool is indicated. Therefore the underlying architecture, the components collaboration and the user interface must be clarified.

3.1 CANDY’s architecture nowadays

Is CANDY a pure web application? The key reason for choosing a web application architecture is the ability to update and maintain web applications without distributing and installing software on clients. This reason is not given for CANDY. One aim of CANDY is the integration of existing tools [LGP+04] and as most of them are not web applications, one has to install at least the "existing tools". One example is, for instance, the *Ekahau Site Survey* program recommended in [Haa05], which is a desktop application.

Thin-client applications, using the request-and-response model, require more networking capability to ensure optimal interaction performance which is demanded for a graphical design utility. Since the deployment and maintenance problem that caused the shift to thin clients is solved by component mechanisms and advanced deploy features, a fat-client architecture should be considered.

The use of a web application architecture for CANDY may not only be un-handly but may even be disadvantageous. The NS2 simulator, for example, needs a lot computing time, even for simple tasks [Gru03]. If CANDY is a web application, then the NS2 simulator will be installed on the CANDY server. Hence one does not take advantage of the "natural" distribution of clients and their computing power.
[Zho04], [Grü04] and [Sch03] use the approach of a web application design. For all other parts of CANDY this is not the case.

**Is CANDY a client-server application?** Once again this statement cannot be verified for every part. Indeed there is no module acting as a client-server application today, not even the Candy Client developed in [Zho05], who proposed this architecture because an essential part of the Candy Client is still missing.

**Is CANDY a batch processing application?** A batch processing application is the sequential execution of a series of programs ("jobs") on a computer. The design approach described in [LGP04] can be seen as a batch processing architecture. "Jobs" are different viewpoints and the sequential execution is given by fulfilling a viewpoint. Furthermore the series of programs are several editors and reporting tools like the "BillReperter" or "RuleChecker". Data transferred in a NDML file. In [LGP04] a reason is given for this approach: it allows CANDY to integrate third party applications like Microsoft Visio [Sch02]. However, at the moment the NS2 simulator module as described [Gru03] is the only CANDY module which uses this way to integrate a third party application.

**Is CANDY a Integrated Design Environment?** In [LGP04] and [LGSW05] was stated that one day CANDY will be an Integrated Design Environment. An Integrated Design Environment combines several programs within one user interface so that it appears like one application. This user interface even behaves like one application though the underlying process is much more complicated. Anyway, to a user of nowadays CANDY it will not look like this: the user has to handle different applications, such as browser, different editors and the user has to copy NDML files between the applications, upload and download them and so one as the only integrating components are the NDML files used by all modules.

CANDY also uses a lot of different programming languages (namely Java, C++, OTcl, Python) and different underlying operating systems (Linux, Microsoft Win-
The tricky part is, that some modules run only on Windows, whereas others only on Linux.

**Is CANDY an application suite, like an office suite?** Presumably this may be the best description of CANDY nowadays. One has different applications for different tasks, and they are loosely connected via NDML files. Like a word-processor for writing documents and a spreadsheet program for carry out some calculations, CANDY offers a Net Editor for creating net topology descriptions and a NS2 simulator interface to run some simulations.

Typical components of an application suite are able to communicate with each other on a higher level, with a consistent user interface between the components. Communication on a "higher level" in CANDY is at least possible through the interchange of NDML files, but a consistent user interface is not given.

**Is CANDY a single application?** To integrating third party applications was always an important aspect, so that there are only some few works, which do pay few attention to this point ([Sch03] and [Zho05]).

A summary is given in table 3.1.

### 3.2 Components related issues

#### 3.2.1 Editors for a cross document model

As mentioned in [LGP+04] CANDY uses different design-stages, named Viewpoints. Every Viewpoint defines a set of pre-conditions and a set of post-conditions. The set of pre-conditions have to be performed to enter the design-stage. The pre-conditions are often implicit, e.g. the cost viewpoint need some object like devices and cables to assign costs. Figure 2.5 show the viewpoints and their relationship towards each other.

After all viewpoints are fulfilled a Reporting Tool is invoked to gather all data of the NDML and to prepare the documentation for the given project.
### 3 Design

#### Related Work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[Sch02]</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>[Gru03]</td>
<td>(all possible)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>[Sch03]</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>[Fäl03]</td>
<td>(all possible)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>[LGP⁺04]</td>
<td>P</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Kaf04]</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Zho04]</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>[Gru⁺04]</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>(X)</td>
<td></td>
<td>(X)⁴</td>
</tr>
<tr>
<td>[LGV⁺05]</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Haa05]</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>[LGSW05]</td>
<td>(P)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Fah05]</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>[Zho05]</td>
<td>(I)⁵</td>
<td>(X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>this work</td>
<td>(P)</td>
<td>P</td>
<td>(P)</td>
<td>(P)</td>
<td>P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⁴ could be classified as batch processing
⁵ the prototype do not use a server, but to use a server is planned

Table 3.1: Architecture design and application integration pattern - 'P' stands for architecture proposal, 'I' indicates that some implementation was done, items in parentheses are assumptive or treated as alternative consideration.
Unfortunately, handling a cross-document model as described in [LGP+04] can be very defective. So, for instance, it might be necessary to consider what will happen if one user goes back one or more design stages and change the content. In one scenario the user might, for instance, delete a device. In such a case, depending on the context, all elements related to that deleted device have to be either updated or deleted as well. If this there was, for instance some cost item attached, it may well be that this cost item would be removed or set to zero. Other elements could be treated different.

A possible solution to this problem could be: CANDY uses a set of editors working together as a unit on a unified model. If one object is changed, it sends a message to all objects which have been registered as listener to that object. The message has to contain information about the operation, which was executed. According to that information all listeners act in conformity with the executed operation, e.g. update their own state or delete themselves. Under this operation a cost object for example would be removed form a list and an IP address could set its state to unused.

This a typical object orientated design pattern of observer and listeners as described in [GHJV95].

3.2.2 RuleChecker

Since the first CANDY related work [Sch02], the RuleChecker idea has existed. This is an appropriated concept to keep a model in sync and consistent, but there exist no RuleChecker in nowadays CANDY.

If it will be only a XML Schema, then the RuleChecker could determining if the NDML is valid XML and could not handle more complex rules, as follows: "A NIC can have one or more IP addresses assigned, except broadcast or unicast addresses", "All IP addresses must be unique world wide , except if the IP address is the localhost-address or ..." in that case it has to be followed some statements for local addresses, addresses behind a firewall and so on. Furthermore it might be necessary to express complicated condition, e. g. "all network devices behind the firewall XYZ".

Not mentioned is if the RuleChecker supports "warnings". A device which is not connected to the network may be a forgotten device or may be an intention on purpose
of the designer to show that this device is isolated and not part of the network. So a "warning" to the user could be helpful. Furthermore, if the designer acknowledges this warning, this state becomes a fact. If one accidentally connects this computer to the network again, this should produce an other warning message, indicating that this device should not be connected.

CANDY is supposed to have a more sophisticated Rule Checker as a XML validator would be.

### 3.3 Cost accounting related aspects

Cost accounting is the process of tracking, recording and analyzing cost associated with the activity of an organization, where costs are defined as "required time or resources". Costs are measured in units of currency by convention, but could also be tracked in other units, for instance in working hours. Cost accounting is one method of a company’s accountancy.

Billing is the process of sending an invoice to customers and is therefore important for companies, designers or consulting engineers.

Two major target user groups with an interest in a billing or cost account facility in CANDY can be considered:

- Consulting engineers designing computer networks
- Companies, operators and accountant, which want to monitor a network and its costs.

Since a invoice for planning and installing a computer network is just an entry in an organizations accountancy, in this work no distinctions are made between the mentioned groups.

Apparently the IT infrastructure of a company is only a small part in the company’s cost accountancy and cost accountancy accounting, which covers only a small part of a company’s business activities. Management accounting deals with the use of
accounting information such as cost accounting and financial accounting to assist
manager within organizations.

CANDY does not model all IT cost locations. Some additional aspects of a com-
pany’s IT infrastructure, which are not taken into account, are given as follows:

- Crew labor cost, rental fee, etc.
- Internet services like database, WWW and FTP servers, domain-controllers,
  DNS-services, etc.
- Advanced online services like payment services, online shops, sales and customer
care, call-centers, content management and networking, etc.
- Security and safety aspects: online backup services, consulting and train-
ing, administration service, firewalls, spam- and virus filter software, software
updates, patches, software migration, etc.
- Running costs for supply and equipment: USV, air-condition, racks, cabinets,
screwdriver, gym shoes and a coffee maker for the administrator
- etc.

Furthermore, the requirement for a billing and cost accounting tool changes during
the network life cycle (for further details see table 3.2).

Additional designers or a consultant engineers needs a lot of information to acquire
a detailed cost analysis. Something CANDY is not aware even if the object belong into
the domain of network design. For instance devices and medias for extreme situations
(fire proof, acid-proof, extreme shielded) can not be modeled with all needed detail
information in nowadays CANDY.

The billing tool proposed in [Sch02] uses a product database with available devices,
medias and NICs linked with detailed information and prices. [Sch02] noted, that the
estimated effort for this approach is high. This database only regards materials and
### Table 3.2: Various aspects of a cost and billing tool.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>by or rent a network, outsourced services,...</td>
</tr>
<tr>
<td>Preliminary planning</td>
<td>Total Cost of Ownership (TCO)</td>
</tr>
<tr>
<td>Investment Plan</td>
<td>estimated Total Cost of Life cycle (TCL) etc.</td>
</tr>
<tr>
<td>Maintain</td>
<td>real-time cost analysis, e.g. printer and traffic counting,</td>
</tr>
<tr>
<td></td>
<td>operation and maintenance services,</td>
</tr>
<tr>
<td></td>
<td>remote technical assistance costs,</td>
</tr>
<tr>
<td></td>
<td>training costs etc.</td>
</tr>
<tr>
<td>Enhance and update</td>
<td>estimated migration and update cost,</td>
</tr>
<tr>
<td></td>
<td>updated TCO and TCL estimations</td>
</tr>
<tr>
<td></td>
<td>advance replacement</td>
</tr>
<tr>
<td></td>
<td>training costs etc.</td>
</tr>
<tr>
<td>Deactivation</td>
<td>scraping and dispose costs</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>

In the accountancy theory exists no uniform way how cost accountancy is done. [Olf05] advices a lot of different cost accounting methods. Beyond theoretically meditations, billing and cost accounting tool can not consider all real-life circumstances: for instance in Germany a lot of ADSL products are offer with a slight variance of options, like city areas discount, gamer option (fast-path), including a free router device, etc. Even a detailed model could not summarize all options.

### 3.4 Conclusion

Summarized there are two major difficulties related with a billing and cost accounting tool for CANDY:

---

1 In Germany it would be necessary to support the HOAI, as long as all construction site activities charged at the rate determined by the HOAI. For instance, T-Systems - a leading network technology supplier in Germany - use the HOAI to charge consulting services.
3 Design

- **Application Architecture** - the integration of a billing and cost tool could be done in several forms: in frame of list driven (or batch) application, as a extension to single applications with export and import facilities or as part of a application suite.

Furthermore the distributed "nature" of the CANDY architecture differs a lot. It ranges from single standalone application with no access to a server till a thin client web application.

A billing tool would need access to all available information, including the NDM, building details and simulation results. Due the lack of a unified model access a user friendly integration is not yet possible.

Also important parts of the framework are missing or not ready to use, in particular the the Documentation tool and the RuleChecker.

- **The Complex Topic of the Financial World** - CANDY can not consider all real life requirements, all financial rules and all related data.

To solve the application architecture problem a new approach is considered: a extensible integration platform with a extensible model and unified model access for all tools and editors. This platform is called *CANDY Platform*. The model should be easily accessible and adaptable in a object-oriented manner. The design of this platform is described in detail below.

The design of a billing and cost accounting tool and its integration into the Candy platform are described in detail on page 25.

### 3.5 Candy Platform

The design of the CANDY Platform is driven by following points:

- The domain of modeling networks is complex and has a need for sophisticate editors with rich and powerful GUIs. Also there is a need for working offline and take advantage of local processor time, for instance to run NS2 simulations.
This rich client should be supported by a server, providing resources and information about the client’s network. The server also is a central station for all agent programs monitoring the network. Both, the server and the client should be extensible in that manner that one can easily add, remove or replace whole parts of the application. Therefore it makes sense to distinguish from a platform and its extensions.

The platform should offer the run-time NDM to all its extensions - so the extension do not have to take care about serialization, communication or handle the difficulties that comes with the distributed character of a cross-document NDML file set.

Likewise the opportunity is provided by the platform to extend all other parts of the CANDY platform, for instance like the help system.

Consequently the NDM should be extensible in order to allow extensions to add new model elements or new behavior as needed.

Figure 3.5 and 3.5 offer a closer view onto this concept.

### 3.5.1 Eclipse RCP

As the basic framework (or platform) of the CandyBox, Eclipse RCP is used.

Eclipse, is well known as an open source framework of building Java application. But behind the popular IDE is a generic platform for running applications, the Rich Client Platform (RCP). And coincidental the IDE is such an application. The purpose of Eclipse and it’s RCP is to provide a highly integrated tool platform.

Some characteristics of a RCP based Application [ML05]:

- **Components** - The RCP includes a component model. RCP based systems are build by combining *plug-ins*. The plug-ins can be shared among RCP applications and have a version number. The same plug-in can be installed with different version side by side and you choose which version you want. Everything
Figure 3.1: CANDY Platform design overview. The user download and start the CandyBox with a Java Web Start able web browser. The CANDY Box is the rich client. It offers a user interface to manipulated the NDM, to create reports and so one. The CANDY Box itself uses the CANDY Server to gain information about the network, to download resources or to verify changes the user made. The user can also communicate with the server via the browser interface. It might be that the server allows the user to start or stop services or the server provides some basic status reports about the network to the user.

is a plug-in, thus every part of the CandyBox can replaced by other plug-ins.
This makes it easy to develop and maintain editors, views, extensions.

- **Infrastructure** - RCP offers a flexible user interface, help support, context-sensitive help, network updates, error-handling and logging, etc.

- **GUI** - The generic workbench provides overall UI organization and supplies the structures in which an application interact with the user. It’s paradigm is based on editors, views and perspectives. From the user’s standpoint, a workbench window consists visually of views and editors. Perspectives manifest themselves in the selection and arrangements of editors and views visible on the screen.

- **Native user interface** - The Eclipse Standard Widget Toolkit (SWT) provides a graphical user interface toolkit for Java. Unlike Swing or AWT, SWT uses native user interfaces of the Operating System and is therefore responsive, better looking and smooth in handling.
Figure 3.2: Details about the CANDY Platform design. The CandyBox has a typical Model-View-Controller architecture. Here, the model is the NDM. The views are either editors or simple views, which present data to the user. Views, editors and the NDM are independent components running inside the CandyBox. The editors using controllers to modify the NDM. A special controller is the RuleChecker. If the NDM contains a error or a problem then the RuleChecker informs the user. A ResourceManager takes care about the necessary synchronization with a CandyServer or the file system if a CandyServer is not used. The CandyServer hosts a Database (CandyStore) and other components (CandyDishes) within a application server.
• **Operating system independence** - the RCP runs on every Java virtual machine with the J2ME foundation libraries. A RCP application is Java Web Start enabled.

• **Disconnect** - because RCP runs on a local machine, it doesn’t need a network connection. This can be a benefit, e.g. form mobile uses.

• **Scheduler** - RCP comes with a Scheduler and complex and elaborated job managing system. This could be used (besides other usage) to have long running simulation processes with low priority in background not blocking the user interface.

• **PDE** - Eclipse PDE supports the designing, developing, building, debugging, packing and deploying of RCP plug-ins.

For more information see the Eclipse RCP web site, [Bo103] and [ML05].

### 3.5.2 EMF - Eclipse Modeling Framework

EMF is a Java modeling framework and code generation facility for building tools and other applications based on a structured model. Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose. From a model specification, EMF produces a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic Eclipse editor.

With EMF, one can model shared data as XML Schema, as annotated Java classes or as UML model. EMF also generates a Java implementation that persists the data as linked XML files. All generated Tools (editors and helper classes) are Eclipse plug-ins. EMF consists of three fundamental pieces

• **EMF** - The core EMF framework includes a meta model (Ecore) for describing models and run-time support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
3 Design

- **EMF.Edit** - The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets. A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.

- **EMF.Codegen** - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

EMF provides a basic validation service for its generated models. This service seems to be a good starting point to setup the RuleChecker. Additional the EMF group is working on a implementation of the OMG’s Object Constraint Language. Furthermore EMF includes the XML Schema Infoset Model (XSD) project and an EMF-based implementation of Service Data Objects (SDO). This might provide a easy way to communication with the CandyServer in a services oriented architecture manner.

### 3.5.3 SimpleCore and Net Editor

To get started a very basic network design model - the SimpleCore model - is used. This model might be replaced by the NDM model provided by [Häh05]. The example (based on [Ani05]) is modified in order to deploy a simple and yet not full functional net editor (called NetEditor).

Both models, the SimpleCore model and the model used by the NetEditor, are EMF generated models.

The SimpleCore model describes a set of network devices, links and NICs and can be modified either with the NetEditor or a SimpleCore editor. The editors for the SimpleCore model and the NetEditor are shown in figure 5.1 (page 46).
Only one instance of a SimpleCore model exists per project.

Else then in [Zho05] it is possible to have more then one NetEditor model resources. This is necessary in order to have separated logical view of the network, for instance one for every room of a building. Therefore the logical view model of the network is distributed among different files.

For every picture "computer" one adds on the NetEditor’s canvas (and to the NetEditor’s model) the NetEditor adds a element computer to the SimpleCore model. If picture "computer" is deleted, the corresponding SimpleCore model element is also deleted by the NetEditor.

So, the NetEditor is 'aware' of the SimpleCore model, but the SimpleCore model doesn’t know how it’s modified or through which editor.

3.6 Cost editor and the cost model

As described before CANDY doesn’t know what users probably wants. Furthermore real-world situation and data requires flexibility. Therefore Candy’s implementations is done as follows:

The NDM is presented to the user through a unified interface. A cost editor allows to enrich the NDM with financial data. With the help of a business intelligence reporting tool (BIRT) and third party information sources like databases and log files, the user of CANDY should be able to create and to deploy business intelligence reports as required.

The capabilities of BIRT are described below.

Unfortunately BIRT needs table like input data. CANDY’s NDM is a hierarchic structured model and therefore not suitable as raw BIRT input data.

There are three possible methods of resolution to enable BIRT to access the NDM:

- by support of BIRT’s scripting interface,
- via the Open Data Access (ODA) framework for adding custom data access methods to BIRT,
In order to create assertoric reports BIRT combines NDM and other data sources, like databases and log files. For example, a database might contain staff department information and various product prices. The log files might contain data gathered by SNMP agents. Together with the network description provided by the NDM, it is possible, for example, to give a complete monthly cost evaluation report as the network operator demands.

- to use a relational database system. NDM data might be mapped into a RDBMS. This approach enables a wide range of other tools to access the NDM, for instance OpenOffice.

The last method was chosen. Figure 3.3 and 3.4 describes the approach how BIRT is integrated into the CandyBox and how the NDM is connected with BIRT.

The RDBMS should only provide read-only access to the NDM and is not used to save persistent data. Therefore an embedded database without access to the file system is sufficient enough. The embedded Hypersonic SQL database (HSQLDB) is chosen for this task and is described in detail on page 29

### 3.6.1 BIRT - Business Intelligence and Reporting Tools

BIRT - Business Intelligence and Reporting Tools - is a Eclipse based reporting system and consists of two main components: a Eclipse based report designer and a run-time
The NDM is adapt through a Database Adapter (DBA). This DBA is generic and can handle any NDM model element. If a new element is added to the NDM a adapter factory is invoked and a DBA adapter is attached to this new model element. The DBA adapter observe to changes of the model element and insert or update corresponding values of the database (a). BIRT queries the database and generate business reports (b). BIRT can either use the embedded in-memory HSQLDB or a external relational database (c). Since BIRT’s report schema are saved as XML files, a report can also be stored in a CandyStore database (e). To run the reports inside CandyDishes will be a major use case and provides a easy, browser based access to network related information and real-time business reports.
component - the report viewer engine. The report viewer engine is used within the CandyBox, but could also be used within CandyDishes. The use the report designer is only meaningful within the CandyBox.

With BIRT one can create following variety of reports:

- Lists - The simplest reports are lists of data. As the lists get longer, you can add grouping to organize related data together (orders grouped by customer, products grouped by supplier). If your data is numeric, you can easily add totals, averages and other summaries.

- Charts - Numeric data is much easier to understand when presented as a chart. BIRT provides pie charts, line and bar charts and many more.

- Letters and Documents - Notices, form letters, and other textual documents are easy to create with BIRT. Documents can include text, formatting, lists, charts and more.

- Compound Reports - Many reports need to combine the above into a single document. For example, a customer statement may list the information for the customer, provide text about current promotions, and provide side-by-side lists of payments and charges. A financial report may include disclaimers, charts, tables all with extensive formatting that matches corporate color schemes.

BIRT reports consist of four main parts: data, transform of data, business logic and presentation.

- Data - Databases, web services, Java objects all can supply data to a BIRT report. JDBC is supported by BIRT, as well as support for using code to get at other sources of data. A single report can include data from any number of data sources. With upcoming release 2.0 BIRT also supports XPath.

- Data Transforms - Reports present data sorted, summarized, filtered and grouped to fit the user’s needs. BIRT allows sophisticated operations such as grouping on sums, percentages of overall totals and more.
• Business Logic - Real-world data is seldom structured exactly as one need for a report. Reports might could require business-specific logic to convert data into information useful for the user.

• Presentation - Once the data is ready, a wide range of options are offered for presenting it to the user. Tables, charts, text and more. A single data set can appear in multiple ways, and a single report can present data from multiple data sets.

For more information and examples see www.eclipse.org/birt.

3.6.2 HSQLDB - Hypersonic SQL Database

HSQLDB is a relational database management system written in Java. It has a JDBC driver and supports a rich subset of SQL-92, SQL-99, and SQL:2003 standards. The HSQLDB core is a fast, small embeddable engine which offers a in-memory-only tables. For more information see www.hsqldb.org.
4 Implementation

4.1 Resource Manager and the SimpleCore model

As mentioned before, a basic model as temporary NDM replacement was needed to show how tools, editors and the models interact with each other. To keep track of, a simple variation of a network model was used: the SimpleCore model. Also a SimpleCore model extensions, the Cost model, is contributed.

The Cost model is simple. It consists of two elements:

- **Cost** - a *Cost* item assigns to a *EObject* attributes like:
  - *cost* - a numeric value to represent some cost. The ISO 4217 standard was not used in order to allow the user to use to give this value a different meaning. For example this value could be used as a counter.
  - *eObject* - a reference to a *EObject*. This allows to use the cost model with every EMF generated model.
  - *typeOfCost* - a value to group cost items,
  - etc.

- **CostSheet** - a *CostSheet* acts as a container for *Cost* items and could be seen as a higher grouping functionality. Every *CostSheet* is stored in a separated XML file.

To create all models (SimpleCore, Cost model) EMF’s annotated java method was used. As example the abstract class *Device* described in annotated java is shown in the following listing:
### Device.java

```java
package de.tudresden.candybox.ndm.simpleCore;

/*
 * @model abstract="true"
 */
public interface Device {

    /*
    * @model id="true"
    */
    String getName();

    /*
     * @model type="NetworkInterface" opposite="device" containment="true"
     */
    EList getNetworkInterfaces();

    /*
    * @model
    */
    String getDescription();

    /**
    * @model type="EObject"
    */
    EList getObjects();
}
```

All additional information the EMF generator needs are indicated with a `/* @model */` tag. For instance the `@model` tag in line 15 tells the EMF generator to create a bidirectional reference. This means, that all referenced `NetworkInterfaces` are "contained" by a `Device`. Whenever a `Device` is removed from the model all contained `NetworkInterfaces` are removed too.
EMF Resources  The EMF provides Resources in order to serialize EMF models and ResourceSets to group Resources. EMF Resources are stored as XML files by default. The listing below shows a EMF Resource - a SimpleCore XML file:

```
'runtime.simplecore'
1 <?xml version="1.0" encoding="UTF-8"?>
2 <de.tudresden.candybox.ndm.simpleCore:SimpleCoreRoot
3  xmi:version="2.0"
4  xmlns:xmi="http://www.omg.org/XMI"
5  xmlns:de.tudresden.candybox.ndm.simpleCore="
6   "http://de/tudresden/candybox/ndm/simpleCore.ecore">
7 <computers name="PC_2" description="PC for the Boss">
8  <networkInterfaces name="eth0"/>
9 </computers>
10 <routers name="FritzBox" description="">
11  <networkInterfaces name="eth0"/>
12  <networkInterfaces name="eth1"/>
13  <networkInterfaces name="eth2"/>
14  <networkInterfaces name="eth3"/>
15 </routers>
16 ...
```

For example routers entity is shown in line 10. Since a router inherit form the abstract class Device its entity has a containment reference to NetworkInterfaces. This containment reference is expressed as a nested element in a EMF generated XML files.

This file is loaded once in each CANDY session through the NdmManager object. The object itself is a singleton and a reference to the object static instance can be gained through the constant member INSTANCE .

```
NdmManager.java
1 public class NdmManager {
2     public static NdmManager INSTANCE = new NdmManager();
3    AdapterFactoryEditingDomain editingDomain = null;
4     URI ndmFileUri;
```
In the default constructor the `ndmFileUri` member is initialized with the URI of a SimpleCore file. Within this prototype implementation the user can not change this URI, thus always the same SimpleCore file is used and all project share the same model. A full functional CANDYBox should provide a more sophisticated project management. The protocol part of the URI ("platform:/..") indicates that the EMF resource loader finds the SimpleCore file inside the Eclipse environment, in this specific case it is a file within the Eclipse workspace. The workspace is responsible for handling projects and resources in Eclipse. With the Eclipse environment comes a lot of dependencies into the CANDYBox, such as the Eclipse resources management. Since the Eclipse resource management focuses on development concerns it is not suitable for the CANDYBox\(^1\).

A customized resource locator implementation will able EMF to locate and load a NDM file from a CANDY server.

\[^1\text{This is one reason, why the CANDYBox offers a 'Resource Perspective' and a 'Navigator View'}\]
The editingDomain is shared between the different editors and can be obtained by calling getEditingDomain. The editingDomain provides a adapter factory, a command stack and a map for various purposes. The adapter factory keeps a list with adapters, for instance label and icon providers.

The command stack allows to redo or undo a commands, for instance the command to delete a computer. Every editor can add its own specific label providers and other adapters to the editingDomain as needed and uses the shared command stack. The user will recognize it, as being able to undo or redo every operation he made not minding which editor he used for executing this specific command.

```java
protected void init() {
    // Load the Package protocol = http host = null
    SimpleCorePackageImpl.init();

    try {
        // Load the resource through the editing domain.
        //
        resource = editingDomain.loadResource(ndmFileUri.toString());
        resourceSet = editingDomain.getResourceSet();
    }
    catch (Exception exception) {
        System.err.println("init faild," + exception.getMessage());
        return;
    }
}
```

During initialization of the NdmManager, the SimpleCore XML file is loaded through the editingDomain and the SimpleCore model is operational.

```java
public Resource getResource() {
    if (resource == null) {
        try {
            init();
        } catch (Exception e) {
            System.err.println("Fehler beim Laden: ");
            e.printStackTrace();
```
4 Implementation

```java
return editingDomain.getResourceSet().getResource(ndmFileUri, true);
}
```

The function `getResource()` returns the loaded resource, that is to say the loaded model. In this case it is the SimpleCore model and will be the full functional NDM as planned in future.

The following listing shows, how the `CostEditor` uses the shared `EditingDomain`:

```
public CostEditor()
{
    super();
    editingDomain = NdmManager.INSTANCE.getEditingDomain();

    adapterFactory = editingDomain.getAdapterFactory();
    adapterFactory.setAdapterFactory(new ResourceItemProviderAdapterFactory());
    adapterFactory.setAdapterFactory(new CostItemProviderAdapterFactory());
    adapterFactory.setAdapterFactory(new ReflectiveItemProviderAdapterFactory());

    editingDomain.setAdapterFactory(adapterFactory);

    BasicCommandStack commandStack =
        (BasicCommandStack) editingDomain.getCommandStack();
    commandStack.addCommandStackListener(new CommandStackListener()
    {
        ...
    });
```

In the `CostEditor`'s default constructor a reference to the shared `EditingDomain` is obtained.

A reference to the `adapterFactory` is received in line 7 and some adapters needed by the `CostEditor` have been added.
Likewise a reference on a BasicCommandStack provided by the NdmManager is received and a anonymous command stack listener is attached to the command stack (line 16).

To attach the graphical NetEditor to the SimpleCore model another approach was used. Since EMF and GEF editors can not share a command stack or a editing domain. This lack of a common infrastructure for tracking command invocation is described in the Eclipse Bug Report #29939 and #37716.

The NetEditor implementation used here is a slightly changed example provided by [Maj04] and [Ani05]. Figure 5.1 on page 46 shows the NetEditor's GUI.

As in [MDG+04] mentioned a graphical editor build with GEF follows the MVC paradigm described in [BMR+]. According to [Hud03] the vocabulary is a little different: in GEF Views are Figures and Controllers are EditParts.

The NetEditor uses it’s own model, for instance to describe the X and Y canvas coordinates of elements. The basic model element of the NetEditor is called Shape and represents either a computer or a router. The graphical controller of a Shape figure is a ShapeEditPart.

If the user places a new computer or router on the NetEditor’s canvas a new SimpleCore model element, a Shape and its controller, a ShapeEditPart is created. The new SimpleCore model element is added to the SimpleCore model. This is done in two steps. The first step is to create a ShapeCreateCommand object. This object is executed and pushed on the NetEditors command stack in order to allow the user to undo this action. Also a new Device object is created. The Shape object maintenance a reference to it. In the second step the Shape is going to be activated. In this step the Device object is added to the SimpleCore model.

```java
ShapeCreateCommand.java
1 ...  
2 public void execute()  
3 {  
```


The function `execute()` is called - as the name implies - when the `ShapeCreateCommand` is executed. This function calls a helper function in order to create the necessary `SimpleCore` model objects.

On the basis on the kind of `Shape` object a SimpleCore model object is created via the provided `EFactory` (line 15 and 18). A reference to the new created device is set in line 22.

```java
private void createSimpleCoreModelElements(Shape shape) {
    EObject dev = null;
    if (shape instanceof EllipticalShape)
        dev = NdmManager.INSTANCE.getEFactory().createComputer();
    if (shape instanceof RectangularShape)
        dev = NdmManager.INSTANCE.getEFactory().createRouter();
    if(dev != null)
        shape.setEObject(dev);
}
```
If a Shape is going to be activated (that means it becomes visible to the user) the function activate() is called by the editor. As in line 12 showed, the function hookIntoModel(Shape) is called and this Shape is linked to the NetEditor’s model. The function getCastedModel() wraps the inherited function getModel() which returns a POJO and cast it to a Shape Object. Likewise the same procedure is done in line 13 and the Shape is linked to the SimpleCore model.

The reference to the device is read cached via the Shape object (line 24).

```java
if(device != null)
{
    Object root = (Object) NdmManager.INSTANCE.getResource().getContents().get(0);
    if(root instanceof SimpleCoreRoot)
    {
        SimpleCoreRoot simpleCoreRoot = (SimpleCoreRoot)root;
        if(device instanceof Computer)
        {
            // Further processing...
        }
    }
}
```
In line 27 and 28 the SimpleCoreRoot is casted and assigned to a local reference. The SimpleCoreRoot object manages several lists, for each device one. A convenient way to add new devices is shown in line 31 and 31. Finally the ShapeEditPart adds itself to the device’s list of adapters (in line 35) and starts to observe it.

### 4.2 The Database Adapter

As shown in figure 3.4 on page 27 the DBA adapt the NDM and maps NDM data into an embedded in-memory database.

**EMF’s Reflective API** The reflective API of EMF generated models is used by the database adapters to create the database schema. This generic solution allows to adapt all EMF generated model.

The design goal is to provide a DBA as an universal model adapter which is able to reflect a broad set of models. The DBA does not make use of the SimpleCore model or other models discussed before. The only source of structural information needed by the DBA and its adapters are the models themselves. All EMF generated models carry their specific design as additional information. Those information can be gathered through the generated reflective API. The primary sources are the generated EPackages, EClasses and EObjects since every EMF generated model class belongs to a EPackage and inherits the EObject interface.
Due to the use of EMF and its resource implementation to provide a basic NdmManger a set of Resource and ResourceSet adapters and their factories are necessary, but not discussed here.

The DBA itself is constructed by following classes:

- **DatabaseDomainController** - provides a simple management which models should be adapted.

- **DatabaseObjectReferenzMapper** - contains two java.util.HashMaps to map a EObject to a UUID and vis-a-vis.

- **DatabaseObjectReferenzImpl** - a class helping to establish the mapping from a EObject to a UUID.

- **DatabaseObjectReferenzAdapter** - this adapter is added to each EObject and extend the behavior of the adapted object and executes database related actions, like INSERT, UPDATE and DELETE. Therefore the adapter uses the SqlStatementGenerator

- **SqlStatementGenerator** - provides various static functions to build SQL statements for given EObjects or EPackages.

The main problem is to map a EMF model (so to say, a Ecore model) to a database schema. This is done by the SqlStatementGenerator. The SqlStatementGenerator uses constructed by following rules:

1. As long as the integrity is guaranteed by the model, the use of database mechanisms, e.g. primary keys, is not essential.

2. Every EObject has a own UUID. This UUID it used as 'foreign key'. Due to the property of a UUID being uniquely a complicated collision detection is not necessary.

3. Every EObject has it’s own table. The table name is derived from the object’s class name.
4 Implementation

4. all simple attributes like strings or numerical values are represented in database counterparts.

5. references to other EObjects are stored in a separated table with two columns. The first column contains the UUID of the object, the second the UUID of the reference.

6. Everything else is ignored.

Since the SqlStatementGenerator implementation is quite complex because of the used interlaced methods only no description is given here.

When a new model element is added to a resource, the attached DatabaseResourceAdapter is informed by the resource and invokes the DatabaseObjectReferenzAdapterFactory which adapt the model element with a new created DatabaseObjectReferenzAdapter. The DatabaseObjectReferenzAdapter is described below:

```java
public class DatabaseObjectReferenzAdapter implements Adapter {
  ...
  public DatabaseObjectReferenzAdapter( Notifier notifier ) {
    this.eObj = (EObject) notifier;
    dbObjRef = DatabaseObjectReferenzMapper.INSTANCE
      .getObjectReferenz((EObject) notifier);
    this.notifier = notifier;
    // Insert into database
    execute(getInsertSQLStatement(), "insert of " + eObj.eClass().getName());
    execute(getUpdateSQLStatement(), "update of " + eObj.eClass().getName());
  }
}
```

The DatabaseObjectReferenzAdapter constructor takes a Notifier object as parameter and resolves the EObject (line 4. Also a UUID (carried by a DatabaseObjectReferenzMapper object) is cached for further use (line 6). In line 10 and line 11 the EObject is insert in to the database. The execute() function per-
forms the SqlStatements and takes a additional String for debugging purpose. The update in line 11 is necessary in order to update all references.

```java
public void notifyChanged(Notification notification) {
    String err = null;

    if (notification.isTouch())
        return;

    switch (notification.getEventType())
    {
    case (Notification.ADD):
    case (Notification.ADD_MANY):
    case (Notification.MOVE):
    case (Notification.UNSET):
    case (Notification.SET):
        {
            err = "SET";
            System.err.println("SET: " + eObj.toString());
            execute(getUpdateSQLStatement(), err);
            break;
        }
    case (Notification.REMOVE):
    case (Notification.REMOVE_MANY):
        {
            err = "REMOVE";
            System.err.println("REMOVED: " + eObj.toString());
        }
    }
```

The function `notifyChanged()` is called when the adapted `EObject` has changed. The event type is determined in line 20, for instance: adding a new element to a list of the monitored `EObject` is indicate by `Notification.ADD` event or `Notification.ADD_MANY`. As result a update statement is performed.

A little bit more tricky to handle is a `Notification.REMOVE` event. This event is released if a referenced is removed from one of the object’s lists.

```java
    case (Notification.REMOVE):
    case (Notification.REMOVE_MANY):
        {
            err = "REMOVE";
            System.err.println("REMOVED: " + eObj.toString());
```
First all reference tables are updated and related entries are removed.

```java
Object oldValue = notification.getOldValue();
if (oldValue instanceof EObject)
{
    EObject eOldValue = (EObject) oldValue;
    if (eOldValue.eResource() == null)
    {
        // remove from the database
        IDatabaseObjectReferenz dbObjRef = DatabaseObjectReferenzMapper.INSTANCE
            .getObjectReferenz(eOldValue);
        if (dbObjRef != null)
            {
                execute(getDeleteSQLStatement(dbObjRef),
                        "DELETE Child.Object ---");
                // remove the adapter
                for (Iterator it = dbObjRef.getEObject().eAdapters()
                    .iterator(); it.hasNext();)
                {
                    Object object = it.next();
                    if (object instanceof DatabaseObjectReferenzAdapter)
                    {
                        it.remove();
                        break;
                    }
                }
        }
        else
        // TODO handle ELists here
    }
break;
```
The removed object (oldValue) is removed from the database (line 51) and its 
DatabaseObjectReferenzAdapter is released (line 61).
5 Conclusion

In this chapter the results are discussed and compared.

5.1 CANDY Platform

The idea of a CANDY Platform seems to be promising. Based on Eclipse RCP and its open source frameworks like BIRT, EMF and GEF it meets the requirements of CANDY to provide a open and extensible model and various graphical appealing editors and user friendly interfaces. Some screen shots of the CANDY prototype are shown in Figures 5.1, 5.3 and 5.3.

The main disadvantage and counter-argument of using RCP and EMF is the high complexity and long time to familiarize one with the provided platform and mechanism. This prototype could only supplied because the time was to short to solve all problems so every part is quasi uncompleted.

5.2 Editors

Two different approaches were shown to build a cross-document model processed by mixed responsibility editors:

- **Cost Editor** - the Cost Editor shared together with the SimpleModel editor one EditingDomain. The different resource (SimpleCore model and Cost model) and were held together in one ResourceSet. Primary this seems to work well, but after a while several problems emerged, e.g. switch editors throw exceptions. It seems, that the EMF framework and its editors is not designed for that use. A quick solution could not be given.
5 Conclusion

Figure 5.1: The SimpleCore editor and the NetEditor.

- NetEditor - the NetEditor do not share the EditingDomain with the SimpleCore editor. The connection is established via the EMF provided and supported adapter pattern. This concept was stable but not full implemented, like the first approach.

Furthermore: the NdmManager loads only the SimpleCore model in advance. All other resources are loaded afterward. Also the editors provide the controllers for the model. For that reason the model can get easily out a synchronized state if a editor is not open.

5.3 Database Adapter

Besides the problems with a cross-document model, the use of EMF (or similar frameworks) and the use of its reflective API is recommendable: generated Java implemen-
tation with a set of utility functions such as factories and XML writers, a reflective API and lot more accelerate and unifies the development process of CANDY.

The adapter pattern is a powerful concept to add additional logic and behavior to a existing model without touching its implementation. With the help of EMF it is even possible to provide a generic extension for not yet existing models. For instance the SimpleCore model could be replaced by a more sophisticated NDM without loosing the cost model and the CostEditor since the Cost model references only EObject and the generic implementation of the DBA. It is possible to attach the DBA to other model to provide cost functionality, e.g to a software management model.

Figure 5.3 shows the integration of BIRT into CANDY.

A problem of the DBA is, that not all model attributes are supported. Additional implementation is necessary to avoid some collisions in table names which are not solved yet and to reduces needless SQL statements. But as BIRT will support XPath in future releases, the DBA could be become useless.

5.4 RuleChecker

A not considered concept provided by EMF is the built in validation service. In future releases of EMF even a OCL implementation will be part of the framework. In the Candy prototype only the basic validation service is used to check the SimpleCore model.

5.5 XML Database

According to the topic of this thesis the demanded implementation and integration of a XML database was done via the Eclipse Modeling Framework. The database schema can be described in all various formats EMF uses to define models: UML, annotated Java, Ecore model description or XML Schemata. Clients, like BIRT or OpenOffice can access the run-time model through a JDBC connection.

The XML Schema respectively the Ecore XML for all model descriptions can be found on the enclosed CD.
5 Conclusion

Figure 5.2: The layout view of the BIRT editor.
Figure 5.3: The preview view of the BIRT editor.
5 Conclusion

5.6 Localization and internationalization

Sine RCP, EMF and BIRT offer a extensive support for internationalization and localization it was not necessary to provide additional implementation work.
Glossary

**BIRT** Business Intelligence and Reporting Tools - BIRT is an open source, Eclipse-based reporting system that integrates with CANDY to produce compelling reports for both web and PDF.

**CAD** Computer-aided design (CAD) is the use of a wide range of computer-based tools that assist engineers, architects and other design professionals in their design activities. (wikipedia definition)

**CANDY** Computer Aided Network Design Utility

**Cost accounting** Cost accounting is the process of tracking, recording and analyzing costs associated with the activity of an organization, where cost is defined as 'required time or resources'. Costs are measured in units of currency by convention. [http://en.wikipedia.org/wiki/Cost_accounting]

**Ekahau Site Survey** "Is a software tool for IT managers, wireless engineers, and WiFi professionals who are planning, deploying, and troubleshooting 802.11 a/b/g networks. The Ekahau Site Survey offers easy planning, quick site surveys, state-of-the-art visual representation, and advanced analysis, optimization, and reporting features. Site surveys have never been this quick and accurate before - in just a couple of minutes you can make sure that you got yourself fully covered." — quoted from www.ekahau.de/

**EN-50173** Standard defined by the *European Committee for Electrotechnical Standardization*. Is the principle design standard for cabling systems installed within the countries of the European Union. In Germany it is also known as "strukturi-
erte Verkabelung*. The standard describes a hierarchical star cabling system which horizontal (or tertiary segment) cabling, building backbone (secondary segment), and campus backbone cabling (primary segment).

**GoF**  Gang of Four - Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides are known as the *Gang of Four*. They are the authors of the book "Design Patterns: Elements of Reusable Object-Oriented Software". See [GHJV95].

**GUI**  Graphical User Interface

**Jakarta Tomcat**  it’s a environment for Java code to run in a web server.

**MVC**  Model View Controller - a design pattern described in [BMR].

**NDML**  Network Design Model Language - NDML is a XML based problem oriented design language, hierarchical and extensible, able to describe basic standards and organizational areas (campus, building, floor), structured cabling systems, data transmission media among other things for mobile and wireless links, active network coupling devices, network nodes and their functional dependencies and parameters, heuristics about customer-specific user applications and their caused network traffic used for simulation, data export/import interfaces and protocols for compatibility toused software as well as to encapsulate remote calls of design applications [LGP+04]

**NDM**  Network Design Model

**NIC**  short form of network interface

**NS-2**  Network Simulator Version 2. See www.isi.edu/nsnam/ns/

**OCL**  The Object Constraint Language (OCL) is a notational language for analysis and design of software systems. It is a subset of the industry standard Unified Modeling Language (UML) that allows software developers to write constraints and queries over object models. These constraints are particularly useful, as they allow a developer to create a highly specific set of rules that govern the aspects
of an individual object. As many software projects today require unique and complex rules that are written specifically for business models, OCL is becoming an integral facet of object development.

**OMG** Object Management Group (OMG) is a consortium, originally aimed at setting standards for distributed object-oriented systems, and now focused on modeling (programs, systems and business processes) as well as model-based standards in some 20 vertical markets. OMG has created the standard for Unified Modeling Language (UML) and related technologies Meta-Object Facility (MOF) and XML Metadata Interchange (XMI). It has further expanded into Model Driven Architecture (MDA).

**OSGi** The OSGi Alliance is an open standards organization formed by Sun Microsystems, IBM, Ericsson and others. Over the past few years it has specified a Java-based service platform that can be remotely managed. The core part of the specifications is a framework that defines an application life cycle model and a service registry (wikipedia). The Eclipse.org OSGI implementation Equinox is the basic platform of the Eclipse RCP. http://www.osgi.org/

**PDE** Plug-in development environment - a set of views, editors, wizards that makes it easier to build plug-ins for Eclipse.

**PDF** Portable Document Format - is a file format developed by Adobe Systems for representing documents in a manner that is independent of the original application software, hardware, and operating system used to create those documents.

**POJO** Plain Old Java Object - This acronym is used to emphasize, that the object is not somehow special, in particular not a EObject.

**RCP** Rich Client Platform

**RDBMS** Relational database management system - a database management system that is based on the relational model as introduced by [Cod70]
Glossary

**RFC**  Request of Comments - a series of numbered technical documents about Internet related information and standards.

**Tomcat**  see Jakarta Tomcat

**URI**  Uniform Resource Locator - Is a structured character string defined in the RFC 2396. A URI identifies a resource. All URIs start with the *schema part* (or protocol part) and the remainder is the *schema-specific part*. Example give: 
http://www.tu-dresden.de

**UUID**  Universally Unique Identifier

**WLAN**  Wireless Local Area Network. See [RT05].

**XML**  Extensible Markup Language is a general purpose markup language. See www.w3.org/TR/xml1
Bibliography


[GHJV95] Gamma, Erich, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. Object-Oriented Technology. Addison Wesley, January 1995. 15, 52


[Sch02] Schmidt, Peter: Projektierung von Unternehmensnetzwerken und Konzipierung einer Entwicklungs umgebung. TU-Dresden, 2002. VI, 2, 3, 4, 5, 6, 12, 14, 15, 17
Bibliography

[Sch03] SCHRÖPPEL, Peter: Entwicklung einer XML-basierten Arbeitsumgebung für die Rechnernetzprojektierung. TU Dresden, October 2003. Diploma thesis. VI, 5, 6, 9, 12, 13, 14
