

Open source: evaluation of database modeling CASE

(HS-IDA-EA-03-203)

Bassam Othman (a00othba@student.his.se)

*Institutionen för datavetenskap
Högskolan i Skövde, Box 408
S-54128 Skövde, SWEDEN*

Examensarbete på programmet för programvaruteknik under
vårterminen 2003.

Handledare: Adam Rehbinder

Open source: evaluation of database modeling CASE

Submitted by Bassam Othman to Högskolan Skövde as a dissertation for the degree of B.Sc., in the Department of Computer Science.

2003-06-13

I certify that all material in this dissertation which is not my own work has been identified and that no material is included for which a degree has previously been conferred on me.

Signed: _____

Open source: evaluation of database modeling CASE

Bassam Othman (a00othba@student.his.se)

Abstract

Open source software is becoming increasingly popular and many organizations are using them, such as apache (used by over 50% of the world's web servers) and Linux (a popular operating system). There exist mixed opinions about the quality of this type of software.

The purpose of this study is to evaluate quality of open source CASE-tools and compare it with quality of proprietary CASE-tools. The evaluation concerns tools used for database modeling, where the DDL-generation capabilities of these tools are studied. The study is performed as a case study where one open source (two, after experiencing some difficulties with the first tool) and one proprietary tool are studied.

The results of this study indicate that open source database modeling CASE-tools are not ready to challenge proprietary tools. However software developed as open source usually evolve rapidly (compared to proprietary software) and a more mature open source tool could emerge in the near future.

Keywords: open source, CASE, database modeling, DDL-code

Table of contents

1 Introduction.....	1
2 Background	3
2.1. Open source development.....	3
2.2. Closed development.....	4
2.3. Database modeling.....	5
2.3.1. Database implementation	5
2.3.2. DDL-code quality.....	6
2.4. CASE-tools	8
2.4.1. Open source CASE	9
2.4.2. Proprietary CASE.....	9
2.5. Tool evaluation	10
3 Problem.....	12
3.1. Problem description	12
3.2. Problem specification	12
3.3. Expected results	12
4 Research method.....	14
4.1. Introduction.....	14
4.1.1. Case study	14
4.2. Alternative research methods.....	15
4.2.1. Literature analysis	16
4.2.2. Interview	16
4.2.3. Survey	16
4.2.4. Implementation	17
5 Preparing for the quality tests.....	18
5.1. Overview.....	18
5.2. CASE-tools and DBMS	18
5.2.1. Open source CASE-tool.....	19
5.2.2. Proprietary CASE-tool.....	20
5.2.3. Database management system.....	20
5.3. Template data models	21
5.4. Quality tests	21
5.4.1. DDL-consistency.....	21
5.4.2. DDL-functionality.....	22

5.4.3. DDL- and table-readability	22
5.4.4. Clean data.....	22
6. Performing the quality tests.....	23
6.1. Implementation of data models.....	23
6.1.1. PyDBDesigner.....	23
6.1.2. Rational Rose	24
6.1.3. ArgoUML.....	24
6.2. Generation of DDL-files	24
6.2.1. PyDBDesigner.....	24
6.2.2. Rational Rose	25
6.2.3. ArgoUML.....	25
6.3. Results from performing the quality tests.....	25
6.3.1. DDL-consistency.....	25
6.3.2. DDL-functionality.....	26
6.3.3. DDL- and table-readability	26
6.3.4. Clean data.....	27
7. Analysis of results	28
7.1. Analysis	28
7.2. Comparison with related research.....	28
8. Conclusions.....	30
8.1. Summary.....	30
8.2. Contributions	30
8.3. Discussion.....	30
8.3.1. Relevance of the results	30
8.3.2. Research method	31
8.3.3. Problems with the tools.....	31
8.4. Possible future work	31
References.....	32
Appendix A1: Template data model 1	34
Appendix A2: Template data model 2.....	35
Appendix B1: Steps involved in creating model and generating DDL- code in PyDBDesigner	36
Appendix B2: PyDBDesigner data model 1	38
Appendix B3: PyDBDesigner data model 2	39
Appendix B4: PyDBDesigner DDL-file 1	40

Appendix B5: PyDBDesigner DDL-file 2	43
Appendix C1: Steps involved in creating model and generating DDL-code in Rational Rose	46
Appendix C2: Rational Rose data model 1	47
Appendix C3: Rational Rose data model 2	48
Appendix C4: Rational Rose DDL-file 1	49
Appendix C5: Rational Rose DDL-file 2	54
Appendix D1: Steps involved in creating model and generating DDL-code in ArgoUML (and um2sql).....	57
Appendix D2: ArgoUML data model	60
Appendix E: Reverse engineering in Rose	61

1 Introduction

Traditional development is motivated by profit and the source code is usually only available to the original developers; the developed software is referred to as proprietary software. With open source development the core of the system (first version of the system, with basic functionality) is spread on the Internet, so that anyone can freely read, modify and redistribute the source code. This allows the system to evolve rapidly through a large number of developers working on it simultaneously, which is also believed to contribute to higher quality with respect to reliability (Stamelos et al., 2002, p. 44).

Proponents of open source claim that the software is of high quality, since the number of developers reviewing the code is large (Stamelos et al., 2002, p. 44). There is however little empirical evidence supporting these claims (Stamelos et al., 2002, p. 45). Open source software (OSS) is, as previously mentioned, distributed free of charge which also could be a reason to use it.

Opponents of open source claim that the produced source code is unreadable and impossible to maintain (Stamelos et al., 2002, p. 56). The open source process is also not well defined, which according to McConnell (1999, p. 8) contributes to lower quality. Stamelos et al. (2002, p. 58) furthermore concludes that quality of the source code produced by the open source community does not reach industrial standards, but is subject to improvements. The code-quality is however better than opponents to open source might expect (Stamelos et al., 2002, p. 56).

Open source CASE-tools are a subset of OSS, which share the previously mentioned characteristics of other types of OSS (high quality, free distribution etc.).

There exist different definitions for CASE (computer aided software/system engineering) in the literature, such as Pressman's (2000) and King's (1997) definitions:

“Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer aided software engineering (CASE), is established” (Pressman, 2000, p. 19)

“‘Computer Aided Systems Emergence’ would be more apt, recognising that we can plan (or engineer) the technical elements to some extent, but that the ultimate outcome will depend upon the less predictable interplay between stakeholder interests.” (King, 1997, p. 328)

Since this study concerns technical issues, Pressman's (2000, p. 19) definition is adopted. King (1997, p. 328) argues that CASE are tools for the organization and that the definition should include the stakeholders interests, this is however not relevant in this study (since only technical issues are considered).

CASE-tools can be used for different development tasks, such as system design and use case modeling; some CASE-tools support multiple tasks. This study deals with issues concerning CASE-supported database modeling; therefore this is the type of tools under consideration.

CASE-tools are according to Maansaari and Iivari (1999, p. 37) considered vital for extensive software development. Most of these tools are proprietary (traditionally developed, with hidden source code) and are often expensive to purchase. An

1 Introduction

alternative to proprietary CASE-tools are open source CASE-tools, which are freely available (with source code) and are often of high quality (Feller and Fitzgerald, 2000, p. 1).

There exist (as mentioned earlier) conflicting opinions about the benefits and drawbacks of open source software, though it cannot be denied that the open source community has produced some impressive (and widely used) software, such as the Linux operating system (Gallivan, 2001, p. 278). The success of some important types of software (such as operating systems) and the benefits (such as low cost and reliability) justify more research in the area.

Quality of proprietary CASE-tools is usually established (or estimated) through a number of factors, such as the reputation of the vendor or some form of quality certification. Quality of open source CASE-tools cannot be estimated using the vendor perspective, since the “vendor” consists of a large number of developers who are (usually) geographically dispersed; thus there is a need to assess the quality of open source CASE (and other types of OSS) using a different approach; this can be accomplished through research similar to this study.

A number of open source CASE-tools have emerged, such as ArgoUML and Dia, but there has been little research about quality of these tools compared to proprietary tools.

There are a number of quality aspects of the CASE-tools that can be studied, such as cognitive support (Robbins and Redmiles, 2000) and the level of methodology support (Jankowski, 1997). This study reports on another aspect of quality of open source CASE-tools, namely quality of automatically generated database source code (DDL-code) from models developed within the tools.

The report is organized as follows. Chapter 2 contains background information on open source development, closed development, database modeling (and DDL-code quality), CASE-tools and tool evaluation. Chapter 3 describes the investigated problem and expected results. Chapter 4 describes the research method that is used in the study and some alternative research methods. Chapter 5 explains what was done to prepare for the evaluation of the tools. Chapter 6 contains the results of the evaluation. Chapter 7 contains an analysis of the results and comparison with related research. Chapter 8 contains a summary and discussion of the results, and some points to possible future work on the research topic.

2 Background

This chapter introduces open source and closed software development, and important issues concerning these types of development. Issues concerning database modeling, CASE and tool evaluation are also discussed.

2.1. Open source development

Open source development relies on the Internet, which enables programmers and developers around the world to be involved in the development of the same software. The idea behind open source consists of spreading the first version of the system (with basic functionality), usually developed by one or a team of developers, on the Internet and allowing others to freely read, modify and redistribute the source code of the system. The evolution of this type of software is extremely rapid compared to proprietary software (Stamelos et al., 2002, p. 43).

To qualify as open source the software must meet a set of requirements, some of which are outlined in Table 1 (Perens, 1997).

Requirement	Description
Free redistribution	Anybody should be able to use the software as part of their own software, without paying a fee.
Source code	The source code must be available and not deliberately obscured.
Derived Works	Modifications and derived software must be allowed. The resulting software must also be allowed to be distributed under the same license.
License must not be specific to a product	The rights attached to the program must not depend on the program being part of a particular software distribution.
The license must not restrict other software	The software cannot restrict other software that is distributed along with it.

Table 1: Requirements that an OSS must meet

The software crisis is said to have contributed to long development time, high cost and low quality in software products. Open source addresses many of these issues, which made it possible to produce reliable, high quality and inexpensive software (Feller and Fitzgerald, 2000, p. 1).

Open source software (OSS) is considered reliable since it is tested by a large number of people. The reason for this is that OSS is freely available to anyone who wants to use it. This also positively affects the security, since the source code is exposed to extreme scrutiny with problems being found and fixed early (Hansen et al., 2002, p. 467).

According to Stamelos et al. (2002, p. 56) traditional developers believe that OSS has a high probability of producing unreadable code that is impossible to maintain. His study does however conclude that these claims cannot be supported.

2 Background

McConnell (1999, p. 8) claim that the open source process is not well defined and activities such as documentation and system testing are omitted. It is also common that the source code contain different programming styles due to the absence of an agreed standard. One of the most well known attempts to informally define an open source process is the one proposed by Raymond (1998) in “The cathedral and the Bazaar” (the cathedral represents traditional development, while the bazaar represents open source), where he describes principles such as “treat your users as co-developers”, “release early, release often and listen to your customers” and “given enough eyeballs, all bugs are shallow”.

Despite all criticism the open source community has produced some successful software. The Linux operating system is one well-known OSS that is widely used and appreciated among users and companies (Gallivan, 2001, p. 278). The apache web server is another OSS running over 50% of the world’s web-servers (Perens, 1997). The popularity and the benefits, such as cost and reliability, justify research in the area from a software engineering perspective, such as the studies performed by Stamelos et al. (2002), O’Reilly (1999) and McConnell (1999).

An increasing number of developers are realizing the benefits of open source and some organizations are launching their own open source projects. Netscape is one such organization and has created the mozilla web browser. IBM has released the secure mailer (an extension to UNIX sendmail program) as open source; they have also launched the website “AlphaWorks”¹ containing the source code of many IBM products.

Software development has not been dominated by open source development, but rather by traditional development where the software source code is (usually) kept secret; this type of development is explained in more detail in the next section.

2.2. Closed development

Closed development (sometimes referred to as traditional development) is usually motivated by profit to the developing organization and the source code is usually kept secret, the developed software is referred to as proprietary software.

Most organizations use a structured approach towards developing software, referred to as software engineering, to maintain software-quality. Software engineering includes the process and technologies used with the process, such as methods and CASE-tools (Pressman, 2000, p. 18). This is an extensive area and is therefore only briefly discussed here.

Software development follows a set of predictable steps, referred to as the software process. Pressman (2000, p. 18) defines a process as a framework for the steps involved in creating high quality software. The process includes activities such as analysis, design, construction (programming), and management (Pressman, 2000, p. 19).

The use of a software process (as defined here) is not limited to closed development, but is usually not used with open source development. Quality of proprietary software is maintained through the use of software engineering (Pressman, 2000, p. 18) compared to open source software, where high quality is (usually) a consequence of other factors, such as a large number of reviewers.

¹ <http://www.alphaworks.ibm.com/>

Databases are becoming an increasingly important part of most software systems (Hoxmeier, 1998, p. 179). Database development is a type of software development and traditional developers usually also use software engineering with this type of development. One step in developing a database is creating a model of it; this is explained in more detail in the next section.

2.3. Database modeling

There exist a number of notations used to model databases. The entity relationship (ER) model, originally proposed by Chen (1976), is one of them. The ER model has been an important paradigm for conceptual database design since it was introduced in the mid-seventies. One reason for the ER-model's wide acceptance is its simplicity and clarity to express real-world objects and their relationships (Fahrner and Vossen, 1995, p. 213). Another reason for its popularity is the easy mapping into traditional data models, including the relational, the network and the hierarchical model.

There exist a number of extensions to ER, such as "ER+" (Kolp and Zimányi, 2000) and "EER" (Elmasri and Navathe, 2000). The simple ER-notation lacks features such as inheritance, generalization and specialization, and union types (Elmasri and Navathe, 2000, p. 73); the ER-extensions introduce new modeling concepts addressing these issues.

UML (unified modeling language) is another widely used notation in many types of development; it was originally not designed for modeling databases, but it has been adapted to database modeling (Elmasri and Navathe, 2000, p. 93). UML is the product of a number of methods that were used in the late 80's and early 90's, for example methods of Booch, Rumbaugh and Jacobson (Fowler and Scott, 1999, p. 1). There are a number of variations of the notation being used for data modeling, as explained by Elmasri and Navathe (2000), where components such as keys, cardinality and weak entities are modeled in different ways.

The higher level conceptual data model is used to represent details about the database that are close to the way many users perceive the data; while the low level physical data model describes physical details about the database, for example how it is stored in the computer. Concepts used in the physical data model are meant for computer specialist and are not necessarily understood by the database developer. The representational data model lies between these two extremes and provide concepts that could be understood by end users but they are also close to the way data is organized in the computer. There are a number of representational models, including the relational, the network, the hierarchical and the object oriented data model. The network and hierarchical data models are considered obsolete and systems using them are sometimes referred to as legacy systems (Elmasri and Navathe, 2000, p. 25). The object-oriented model is becoming increasingly popular, though the relational model has been widely used and is still used in many commercial DBMS:s (Elmasri and Navathe, 2000, p. 25). The relational model is the one considered in this study, since it is the most widely used model. Elmasri and Navathe (2000) give a more detailed explanation of the concepts of database modeling.

2.3.1. Database implementation

After modeling the database it needs to be implemented and executed on a database management system (DBMS). The implementation process includes specifying conceptual and internal schemas of the database and any mappings between them. In many DBMSs available today only one language is used to define both schemas, the

data definition language (DDL). The file that contains the source code is referred to as the DDL-file. This file is then processed by a DDL-compiler, which is usually part of the DBMS (Elmasri and Navathe, 2000, p. 30).

The SQL-language is the standard language used with relational databases, which is one of the reasons for the success of commercial relational databases (Elmasri and Navathe, 2000, p. 243). This makes it easier to switch the DBMS, since the DDL-code should be executable on the new system (providing that it supports the SQL standard). There are however variations among DBMS, but if the database faithfully follows the standard, the conversion should be much easier (Elmasri and Navathe, 2000, p. 243).

2.3.2. DDL-code quality

A number of studies have been performed dealing with database quality, such as Piattini et al. (2001), Levitin and Redman (1995), and Hoxmeier (1998). The studies deal with different aspects of database quality, such as data model quality (Levitin and Redman, 1995) and database maintainability (Piattini et al., 2001).

This study deals with quality of the DDL-code, which is defined as the level of adherence to a set of requirements collected from previous work in the area. The requirements deal with different issues of the DDL-code, such as readability, correctness and effectiveness. The quality requirements are summarized in Table 2; sampled from Rehbinder (2000, p. 159-163).

Requirement	Description
1. DDL-consistency	The generated DDL-code should describe the same database as the data model.
2. DDL-functionality	The generated DDL-code should be executable.
3. DDL- and table-readability	The DDL-file should not contain any unexpected code or symbols, so that a developer can understand the code as if he had written it himself.
4. Clean data	The use of triggers and stored procedures should be as sparse as possible to prevent high complexity.

Table 2: The quality requirements

The requirements are explained in more detail in the following sections, where different issues concerning each requirement are discussed.

DDL-consistency

This requirement concerns consistency between the conceptual model and the DDL-code, i.e. the DDL-code must describe the same database as the model. There are several issues concerning this requirement, these are explained in Table 3.

Requirement	Description
1.1. Information preservation	One goal of the schema transformation is to preserve the information in the model (Fahrner and Vossen, 1995, p. 219) i.e. the DDL-code must be able to hold the same information as the model.
1.2. Implicit integrity constraints	The model could include implicit integrity constraints, which has to be described in the DDL-code (Fahrner and Vossen, 1995, p. 219).
1.3. Triggers, constraints, stored procedures	Modeled (assuming that the tool supports this function) triggers, constraints and stored procedures should be correctly implemented in the DDL-code (Rehbinder, 2000, p. 159-163).
1.4. Inheritance	If inheritance can be modeled in a direct way, i.e. without any “special solutions” (such as extra tables), it has to be implemented correctly in the DDL-file (alternative ways to implementing inheritance can be found in Elmasri and Navathe (2000, p. 295-299)).

Table 3: Issues concerning DDL-consistency

DDL-functionality

To satisfy this requirement the generated DDL-code must be executable on the target DBMS. There are no special issues concerning this requirement (if the DBMS accept the code, then the requirement is met).

DDL- and table-readability

The DDL-code should not be unnecessarily obscured, i.e. the DDL-file should not contain any unexpected code or symbols. The developer should be able to read the DDL-code as if he/she had written it himself.

There are a few issues concerning this requirement, these are outlined in Table 4.

Requirement	Description
3.1. Comments enhancing readability	Comments could enhance readability.
3.2. Unnecessary commands	The DDL-file should not contain any redundant commands.
3.3. Names	Names (for tables, attributes etc.) in the DDL-code should be consistent with the model.

Table 4: Issues concerning DDL-readability

Clean data

The DDL-code should not contain an unnecessary amount of triggers and stored procedures, to prevent high complexity.

2.4. CASE-tools

There is no agreed definition for CASE and different definitions have been proposed in the literature. Pressman (2000, p. 19) defines CASE as a system for the support of software development. King (1997, p. 328) argues that CASE are tools for the organization and that the definition should include the stakeholders interests. The abbreviation CASE is usually defined as computer aided software/systems engineering.

CASE-tools are considered necessary for extensive software development (Maansaari and Iivari, 1999, p. 37). These tools are designed to help the developer in many ways. They should (among other things) make software development easier and more effective, this is however not always the case. Much research has been done dealing with benefits and drawbacks of CASE. King (1996) performs one such study, where organizational issues are considered. The study could also be focused on technical issues such as the study performed by Post and Kagan (2000). In this study CASE-tools are considered a technical aid for the developer that, in this case, reduces the effort needed to design and implement a database.

There exist different types of CASE-tools, which facilitate different types of development. Some tools can be used for multiple development tasks, while other tools have a specific purpose. The CASE-tools that are considered in this section are partly or mainly built for database modeling.

The tools considered in this study have been chosen with some attributes in mind (given the foci of this study, which is to evaluate DDL-generation capabilities in the tools); the attributes are outlined in Table 5.

Attribute	Note
1. UML and/or ER notation must be supported.	These two notations are commonly used in database modeling.
2. Automatic generation of DDL-code from the model must be possible.	This is an essential attribute for this study, since it is this aspect of the tool quality that is studied.
3. Popularity and use are considered for proprietary tools.	This attribute is considered for the choice of tools that are representative, through a search of companies using them and/or literature analysis.
4. Maturity is considered for open source tools.	It is hard (if not impossible) to assess the popularity of open source tools in the same way as proprietary tools, since "customer lists" are not present. Therefore the maturity (i.e. how far the development of the tool has progressed) of the tool is used as selection criteria.

Table 5: Attributes for selecting CASE-tools

UML and ER notation are commonly used in database modeling (Fahrner and Vossen, 1995, p. 213; Elmasri and Navathe, 2000, p. 93), which is the main reason for using them in this study. The second attribute is necessary since it is that aspect of the tools that is studied. Attribute 3 and 4 are used to choose appropriate tools. The

attributes presented in Table 5 are used to create samples of open source CASE-tools and proprietary CASE-tools, in the following two sections.

2.4.1. Open source CASE

Open source CASE-tools have the same benefits as other types of OSS, such as being distributed freely. Table 6 contains all open source CASE-tools that have been found, with the attributes in Table 5.

CASE-tool	Note
PyDBDesigner	This tool is under development. The current release (version 0.1.3) does however implement important features such as automatic DDL-code generation. The tool is built only for database development and uses the ER model.
Dia	The generation of the DDL-code is handled by a separate tool (a number of tools are available). This tool can also be used with other types of development and support both ER and UML notation.
ArgoUML	The tool does not directly support the generation of DDL-code, but support the XMI-format (XML-based exchange format) for UML diagrams, which can be used as input to an sql code generation tool.

Table 6: Examples of open source CASE

2.4.2. Proprietary CASE

Proprietary CASE-tools have been around longer than open source CASE and are usually expensive. The tools in Table 7 are a small sample of what is available on the market. A search on company and vendor websites indicates a large number of companies using these tools.

CASE-tool	Vendor	Mainly database modeling
Allfusion ERWin Modeler	Computer Associates	Yes
ER/Studio	Embarcadero Technologies	Yes
Rose	Rational Software	No
Visio	Microsoft	No
PowerDesigner	Sybase	No

Table 7: Examples of database modeling proprietary tools

2.5. Tool evaluation

Tool Evaluation can be done at the organizational level where aspects such as cost and productivity are considered. The data could be collected through questionnaires containing appropriate questions to the developer. The data is then analyzed and conclusions are drawn from the results. King (1996) performs a study at this level, where cost, productivity and other organizational issues are considered.

The evaluation could also be focused on technical issues such as the level of methodology support in the tool. There are a number of studies focusing on this aspect, for example Jankowski (1997), and Post and Kagan (2000), which are summarized in Table 8. Another issue is the level of cognitive support, such as the study performed by Robbins and Redmiles (2000) (also explained in Table 8).

Study	Description
Jankowski (1997)	<p>The impact of methodology support, in CASE-tools, on specification quality is investigated.</p> <p>A framework for comparing the level of methodology support in CASE-tools is presented, and applied to two tools.</p>
Post and Kagan (2000)	<p>In this study object-oriented methodology support in Rational Rose (a propriety CASE-tool).</p> <p>The tool is studied through questioning developers on their experiences.</p>
Robbins and Redmiles (2000)	<p>Three attributes of ArgoUML (an open source CASE-tool) are studied, where cognitive support is one.</p> <p>A set of desired design features intended to support the design tasks are explained in the context of ArgoUML.</p>
Stamelos et al. (2002)	<p>This study examines quality of open source software.</p> <p>The researchers examine quality of a sample of OSS using a testing tool. Quality of software is defined, in their study, as quality (which is defined by the testing tool) of source code.</p>

Table 8: Examples of studies dealing with technical issues

Stamoles et al. (2002) perform a technical study focusing on source code quality of OSS (i.e. program source code). The evaluation procedure in their study consists of examining a sample of OSS using a testing tool, where a set of criteria is evaluated. The evaluation procedure in their study is similar to this study; although no testing tool is used here, the DDL-code (database source code) is examined using a set of predefined criteria (which, in their study, is defined in the testing tool).

2 Background

Another technical issue of interest to database developers is quality of databases generated from models (assuming that the tool has support for this) by the tool. Quality of the database includes correctness, for example the database has to be executable and not contain any syntactic errors (error in the language). Another aspect of the correctness is what the source code means, i.e. what is implemented, which must be the same as what the model represents.

3 Problem

There are many conflicting ideas about the benefits and drawbacks of CASE-tools, though most research supports the use of the tools (King, 1996, p. 175). CASE-tools are furthermore considered vital for extensive software development (Maansaari and Iivari, 1999, p. 37).

The market today is dominated by proprietary CASE-tools, which are usually expensive to purchase. Over the past years open source development has attracted more attention and some successful software has been produced this way (e.g. Linux). Open source CASE-tools share the characteristics of other types of OSS, such as free distribution, and could be a cheaper and more reliable (although not always the case) alternative to proprietary CASE-tools.

There are a number of aspects of CASE-tools, which are subject to research. Such areas include technical issues such as the level of methodology support or the level of cognitive support, but also organizational issues such as cost and productivity. This study deals with CASE-tools used for database modeling that also support automatic generation of database from models. The generated database is evaluated and quality (as defined in the next section) is assessed.

3.1. Problem description

This report addresses technical quality issues involving CASE-tools used in the database modeling area. An evaluation of a representative set of proprietary and open source CASE is performed and the results are compared to assess general differences in quality between the two types of CASE-tools.

Quality aspects include adherence to agreed standards (such as UML and XMI), ease of use and reliability aspects (not involving the DDL-code). The focus of this study lies on one aspect of quality; the DDL-code that the tools generate. Quality of DDL-code is defined as the level of adherence to a set of requirements. The evaluation procedure consists of testing and reviewing DDL-code using these requirements.

3.2. Problem specification

This study examines quality of DDL-code generated by an open source CASE-tool, built for database modeling and compares the results with a similar examination on a proprietary CASE-tool.

3.3. Expected results

Stamelos et al. (2002) performs a study on quality of OSS, where they study a sample of OSS with a testing tool. Quality of software is defined, in their study, as quality of source code (which is defined by the testing tool). They conclude that the open source community produces software that does not reach industrial standards, but is better than opponents of open source might expect (Stamelos et al., 2002, p. 56).

According to McConnell (1999, p. 8) the open source process is not well defined, which is also a reason why OSS is expected to be of lower quality than proprietary software. Opponents of open source also claim that open source has a high probability of producing unreadable code, which is impossible to maintain (Stamelos et al., 2002, p. 56).

3 Problem

As indicated there are conflicting views about the quality of OSS. The results from this study should give an indication of the level of quality of open source CASE-tools, designed for database modeling, compared to proprietary CASE-tools.

The results from this study does of course not provide a complete picture of the usefulness of open source CASE-tools, but combined with similar research studying other quality aspects it could influence perceptions of open source CASE-tools.

4 Research method

This chapter presents the research method used to achieve the aim of this study, and some possible alternative research methods. Section 4.1 contains an introduction to research methods (in the context of this study) and a presentation of the case study research method, which is the research method chosen for this study. Section 4.2 presents alternative research methods.

4.1. Introduction

Maxwell (1996, p. 65) identifies 4 main components of a research method:

- **The research relationship with the subjects:** the relationship that the researcher has to the individuals that he/she studies could affect the results.
- **Sampling:** what is selected (individuals, tools etc.) for observation.
- **Data collection:** how information is gathered.
- **Data analysis:** how the data is interpreted.

The first component is not part of this study since there are no interviews and individuals are not studied at all (instead CASE-tools are studied).

Maxwell (1996, p. 70) describes different types of sampling, such as probability sampling (random sampling) and purposeful sampling. Purposeful sampling is used in this study, which means that the samples are chosen deliberately in order to provide valuable information that doesn't exist with other samples. The samples (CASE-tools) in this study were not chosen randomly; instead the attributes mentioned in Table 5 were used to make the choice.

The data in this study is collected through observations of results after executing a set of tests (section 5.4). According to Maxwell (1996, p. 73) triangulation (the use of multiple research methods) could enhance the credibility of the results; this is however considered unnecessary in this study, since the case study research method (alone) provides sufficiently reliable results.

The data in this study was analyzed through a comparison of the results from the examination of the open source tool and proprietary tool.

The case study research method is the research method chosen for this study and is therefore explained in more detail in the next section (along with a motivation of why it is relevant for this study). Alternative research methods are discussed in section 4.2.

4.1.1. Case study

The case study research method enables the researcher to study a phenomenon in its natural setting; this is also the preferred research method for this study. A case study should not be mistaken for a survey where the cases are more extensive. There are three main reasons for using it in this study (Benbasat et al., 1987, p. 370):

- The researcher can study the system in its natural setting and generate theories from practice. The natural setting in this study is defined as using the tools to implement “real” (complete) models (the models in appendix A1 and A2). Theories (quality-comparison of open source and proprietary CASE-tools), in this study, are generated from the experiments (cases), where the two CASE-tools are tested and the results are compared.

4 Research method

- The researcher can answer “how” and “why” questions. The question to be answered in this study falls into the first category, since it is a comparison question.
- The case study research method is suited for areas in which few previous studies have been carried out. There has, as mentioned earlier, not been much research in the area of open source CASE-tools.

A case study examines a phenomenon using information from one or a limited number of cases. The cases could be organizations, individuals, tools or any other units (Benbasat et al., 1987, p. 370). The cases in this study are the studied CASE-tools, where information is gathered through the execution of a set of steps (explained further in chapter 5), including analysis of the output from the generation step (the DDL-code); some background information is also gathered through a literature review (for example the quality requirements used to evaluate the tools).

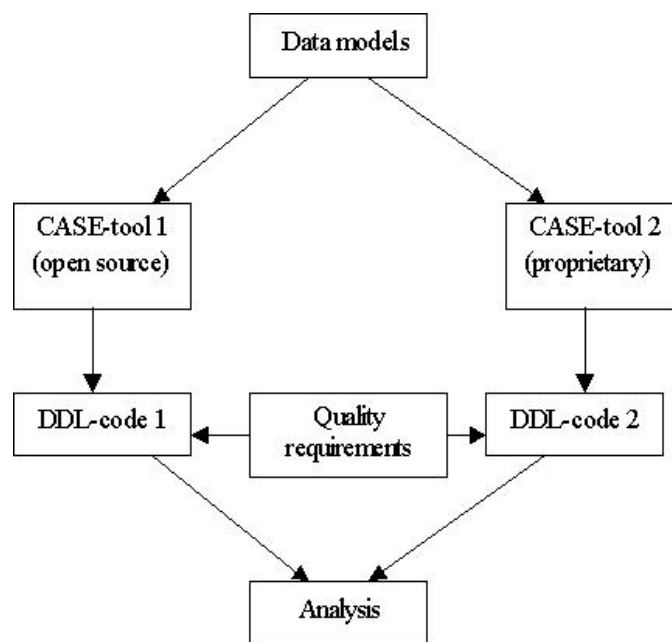


Figure 1: Case study setup

Figure 1 illustrates how this study was planned. The steps involved are:

- Two data models were implemented in each tool (discussed in section 6.1). The models are referred to as template data models and were chosen so that they have all the necessary characteristics making it possible to evaluate all the quality requirements in section 2.3.2 (explained in detail in section 5.3).
- DDL-code was generated from each model, in each tool. This is explained in section 6.2.
- Both DDL-files were evaluated separately using the quality requirements, through the execution of a number of tests (section 5.4).
- The results (section 6.3) from executing the tests were compared and analyzed (chapter 7).

4.2. Alternative research methods

Although the case study research method is the chosen research method for this study, there are alternative methods that could have been used in this study. This section

explains these in the context of this study and gives a motivation to why they are not chosen.

4.2.1. Literature analysis

A literature analysis (referred to by Yin (1994) as a history) is a systematic examination of previous work in the area (published sources such as articles, books etc.), with a specific goal in mind. Literature analysis should not be confused with literature review, where the goal is to get familiar with a specific subject.

Literature analysis could be suited for studies where the researcher has little control over actual behavioral events (Yin, 1994, p. 8). Although a literature analysis could deal with current research questions, it is common to use this type of research method when there is no access to current material and the researcher has to completely rely on past findings (Yin, 1994, p. 8).

A pure (study centered on) literature analysis would be inappropriate for this study, since there has been little work in the area and it would be hard (if not impossible) to get data for generating useful results.

If there were enough material, a pure literature analysis could be performed through examining this material and focusing on researchers' perceptions of quality of open source CASE-tools and proprietary tools, and then perform a comparison between these.

4.2.2. Interview

This study could be performed using interviews, the interviewees would be developers using open source CASE-tools and developers using proprietary CASE-tools (or both). The interview questions would concern experiences that developers have with these tools when using them to generate DDL-code.

Interviews can be performed in different ways, for example open (similar to discussion) or closed (predetermined set of questions) interview (Berndtsson et al., 2002, p. 61).

The use of interviews in this study could get problematic when trying to find developers using open source tools, since there is little documented about the users of these tools. Another issue is that most large organizations (as explained in chapter 2) use proprietary tools, which could also make the study more problematic. These are the main reasons for not choosing this research method.

4.2.3. Survey

This study could have been formed as a survey with questionnaires sent to developers using both proprietary and open source tools (as with an interview). The questions could have been formed in a similar way as with interviews, but it should however be possible to answer them in a more direct way (since the amount of questionnaires is usually large).

A survey relies on data collected through questionnaires or closed interviews. The data is then analyzed using statistical methods. This type of research method is particularly suitable for studies where the number of respondents (cases) is large. One advantage of surveys is that the researcher can reach a large number of people in a short time and with relatively limited resources (Berndtsson et al., 2002, p. 64).

4 Research method

The problems discussed with interviews (previous section) are also present here and are also the main reasons for not choosing this research method for this study. Another reason for not choosing this research method is that it could be complicated to analyze the questionnaires due to the nature of this research question (which is to study quality in the tools), since the answers to some (if not most) questions would be based on personal experience and (many) variations among a large number of samples are likely to occur.

4.2.4. Implementation

Some studies consist of developing new solutions to problems such as a method, procedure or algorithm; that have some advantage over existing solutions (Berndtsson et al., 2002, p. 65). It could be appropriate (or necessary) to implement this solution in order to prove that it has the alleged advantages (e.g. when making a proof of principle).

If resources were not an issue and if the researcher had enough experience, this study could be formed as an experiment in the following way (with minor modification to the research question):

The open source CASE-tools that exist today are either immature or does not directly support generation of DDL-code (as explained in chapter 2). A solution to this could be to find a mature open source tool (such as ArgoUML) and download the source code (which is freely available) and implement this functionality (which would be the new procedure that the study develops) in the tool. The tool could then be spread again (as open source) and users might comment on its functionality and source code (the solution could also be accepted by the original developers, which in turn could bring more trust to the “new” version of the tool). The tool is also tested and compared to a proprietary tool (with respect to quality requirements).

This research method does not seem suited for this study, since only the first step (understanding the original source code) would require a lot of resources and could be hard accomplish, even for experienced programmers (if the source code is not written in a structured way).

5 Preparing for the quality tests

This chapter presents what was done to prepare for the quality tests. Section 5.1 contains an overview of the steps involved in achieving the goal, section 5.2 presents issues concerning choosing CASE-tools and DBMS, section 5.3 presents the template data models (used as a blueprint for the two models created in each tool), and section 5.4 presents the tests performed to evaluate the quality requirements.

5.1. Overview

To prepare for the quality tests a number of steps were performed:

- Selection of two appropriate tools (one proprietary and one open source), considered representative for the two types of CASE-tools (section 5.2). An appropriate DBMS was also chosen for the testing of the DDL-scripts (section 5.2).
- Selection of models that capture (implement) all the requirements defined in chapter 2.3.2, referred to as template data models. These models did not need to be created using a CASE-tool (section 5.3).
- Implementation of the models selected in section 5.3 was performed in both the open source and the proprietary tool (and one model in a third open source tool).
- Planning of the tests to evaluate the quality requirements defined in section 2.3.2 (section 5.4).

5.2. CASE-tools and DBMS

It is hard to choose tools that are representative for the two groups (since there is a huge amount of tools available). The choice was based on attributes such as popularity and experiences among the organizations.

The tools in Table 6 and Table 7 are samples of CASE-tools with the necessary attributes, which are ER and/or UML notation support and DDL-code generation capabilities. The popularity of the proprietary tool was assessed through an investigation of the number of users (organizations); while very little or no user information is present about the open source CASE-tools. The open source tool (s) was chosen (from the samples in Table 6) based on its maturity, i.e. how far the development of the tool has progressed. The choice of open source tool was problematic due to poor functionality (discussed further in section 5.2.1).

The most important requirement put on the DBMS is adherence to the SQL-standard; since the SQL-language was used in the DDL-code generated by the CASE-tools (and it is also widely used). Other factors that could affect the choice are:

- Availability. E.g. if a fully featured evaluation version is available or if the DBMS is open source.
- Reliability. Which is an issue subject to much discussion and should be evaluated separately. This is however beyond the scope of this study and previous work or user opinions are considered instead.

5.2.1. Open source CASE-tool

The maturity of the open source CASE-tools are assessed through investigating the “vendor” (tool homepages) release notes and some exploratory testing of the tools. The maturity is defined by factors such as if bugs appear during initial testing, the number and severity of bugs that are reported at the tool homepage, and the level of functionality (e.g. how many modeling structures are supported).

The examination of the tools indicates that ArgoUML² (version 0.12) is the most mature tool of the three tools presented in Table 6. ArgoUML uses, as the name implies, UML notation (only). The investigation of the tools was focused on their modeling capabilities (and not the generation aspect), in that aspect ArgoUML appeared superior to all the other tools in Table 6. The model was successfully created in this tool, but the generation (which is performed with a separate tool) was a total failure (explained further in section 6.2.3). This made it impossible to use ArgoUML in this study and another tool was chosen (PyDBDesigner version 0.1.3).

ArgoUML is introduced (since it has influenced this study) in the next section, while PyDBDesigner is introduced in the section after.

ArgoUML

ArgoUML is an open source modeling tool written in Java (an object oriented programming language) using the UML-notation. ArgoUML does not directly support generation of DDL-code, but rather through a separate tool using the XMI-format, which complicates the DDL-generation step. Despite this shortcoming (which is not present in only one of the tools in Table 6) ArgoUML was tried, since it is considered significantly more mature than the other tools (with respect to the modeling capabilities).

The XMI (XML metadata interchange)-format is an XML based exchange format between UML based tools; ArgoUML version 0.12 uses XMI 1.0 for UML 1.3, as saving format.

The DDL-generation is performed with a small Java-based open source tool called “uml2sql” (version 0.8.0)³, which takes an XMI-file as input and generates DDL-code. The tool is still under construction and is missing some functions, such as support for stored procedures.

To be able to run ArgoUML and the generation tool, a “Java Virtual Machine” (JVM) needs to be installed (version 1.4.0 is used in this study); this is freely downloadable from the vendor homepage⁴ (although not open source). Since ArgoUML runs “on top” of a JVM it requires a lot of computer resources. This could become irritating, since it executes slowly even on modern computers⁵.

PyDBDesigner

PyDBDesigner⁶ stands for “Python database designer” and is an open source database-modeling tool written in python (an interpreted object-oriented programming language).

² <http://argouml.tigris.org>

³ <http://uml2sql.sourceforge.net/>

⁴ <http://java.sun.com/>

⁵ The computer used in this study: Intel P4 2,4 GHz, 512 RAM, 40 GB HD

⁶ <http://pydbdesigner.sourceforge.net>

5 Preparing for the quality tests

PyDBDesigner requires this software to be able to execute:

- The Python interpreter
- wxPython, a GUI toolkit for python

The installation of the above software should work on both windows and Linux (which is the operating system used in this study with the tool). PyDBDesigner do not need any installing (it is executed directly).

The tool is, according to the creators, under heavy development, which these issues also indicate:

- Relations could not be modeled directly, instead foreign keys are added and the relations are then “automatically generated”.
- No support for triggers, constraints and stored procedures.
- Error messages are not displayed in the tool (for example as “popup-windows”), but instead they are displayed in the background terminal (shell in Linux or DOS-window in Windows).
- Some common user mistakes (such as not specifying a type for an attribute) could cause other errors (such as attributes being added twice).
- The tool does not display any warnings if a file has not been saved, when closing; which could cause loss of data.

This is however understandable to a certain extent, since the tool is in its early stages and not widely used (if it were, there would be more people reviewing the code and more people would be joining the development team).

One advantage that PyDBDesigner has over ArgoUML is that it is possible to generate DDL-code within the tool (no extra tool is needed).

5.2.2. Proprietary CASE-tool

The chosen propriety CASE-tool is Rose (version 2002.05.00), from Rational software (now owned by IBM)⁷. The number of users of this tool is large (according to the vendor). The tool can be used for development tasks other than data modeling; a module (“sub-tool”) is used for database modeling called “data modeler”.

Rational Rose uses UML notation (only) and the DDL-generation is performed within the tool. Since data modeler is a module used with Rational Rose, it cannot be used as a standalone application. This could be perceived as a disadvantage, since Rose is very expensive to purchase and some users may not require other capabilities.

5.2.3. Database management system

The chosen DBMS is Borland Interbase⁸ version 6.0.1.0 (using IBConsole version 1.0.1.340). This is a proprietary DBMS, but an evaluation version can be freely downloaded. A full version is however available to the researcher and used in this study.

⁷ <http://www.rational.com/>

⁸ <http://www.borland.com/interbase/>

The DBMS implements the SQL standard (according to the vendor), which is the most important requirement, set on the DBMS for this study. The reliability of this DBMS is considered acceptable, since it is widely used.

5.3. Template data models

The models (appendix A1 and A2) have been taken from previous related work, namely Kolp and Zimányi (2000, p. 1070), and Shoal and Shiran (1997, p. 310). The used notation in both models is an extension of the ER-model; referred to as ER+ (model 1) and EER (model 2) by the authors. The models (with some additions) have the necessary characteristics making it possible to evaluate the requirements in section 2.3.2, these are:

- Attributes in tables are present.
- Attributes in relations are present.
- Relations: many-many (m:n), one-many (1:n) and one-one (1:1), are implemented.
- Relations: recursive and in-between tables are implemented.
- Inheritance is implemented

To be able to fully evaluate the quality requirements some additions to model 1 (appendix A1) have been made:

- A multi-valued attribute “friends” was added to the “client”-table.
- A recursive relation “supervisor, supervisee” was added to the “driver”-table.

These additions are one of several ways to complete the model.

It is likely that there exist a large number of alternatives that also have the above characteristics. The choice among these models is based on simplicity, clarity (which is also present in other models) and used notation (ER or UML). The chosen models are easy to understand, since (among other things) keys, relations and cardinality are clear; the models use ER+ and EER notation, which is acceptable.

Stored procedures, triggers and constraints have not been modeled in the template data models. These are not supported by the open source tools and can only be created in the proprietary tool. This makes the proprietary tool superior in that aspect and was not used in the evaluation of the tools (since it would not be possible to compare the results).

5.4. Quality tests

This section presents the tests performed on the DDL-code to evaluate the level of adherence to the requirements presented in Table 2. The requirements are explained in detail in section 2.3.2.

5.4.1. DDL-consistency

Information preservation

This requirement is evaluated through making sure that the DDL-code can hold the same data as the model. This is done through reviewing the code with the following aspects in mind:

- All attributes present in the model have to be implemented in the DDL-code.

5 Preparing for the quality tests

- All relations have to be correctly implemented (foreign keys have to be correct) in the DDL-code.
- Tables that inherit properties from other tables have to be correctly (with inherited properties) implemented in the DDL-code.

Implicit integrity constraints

This requirement is evaluated through making sure that all implicit integrity constraints are present in the DDL-code. Implicit integrity constraints include:

- Foreign key constraints: some foreign keys should have constraints on them to correctly implement a relation in the model (e.g. mandatory participation).

Triggers, constraints, stored procedures

This requirement is evaluated through making sure that modeled triggers, constraints, and stored procedures are correctly implemented through reviewing the DDL-code.

This is however (as mentioned earlier) not tested in this study, since there is no support for it in the open source tool.

Inheritance

The evaluation of this requirement is performed through examination of tables that are modeled to inherit properties from other tables and checking the code so that these are implemented correctly. There exist five tables (Passenger, Agency, Freq_trav, Ordinary and Special) in the template data model that inherit properties from other tables. There are several ways of “translating” inheritance to other structures, these include (Fahrner and Vossen, 1995, p. 222):

- An ordinary relation (the supertype primary key is modeled as a foreign key in the subtype).
- Supertype and all subtypes are represented in a single relation.
- Inheritance can also be translated into weak entities (which is proposed by the documentation of Rational Rose).

Inheritance was modeled using weak entities in Rose, since it was not possible to model inheritance directly.

5.4.2. DDL-functionality

The evaluation of this requirement is simple: the DDL-code is executed on the DBMS described in section 5.2.3. If the DBMS does not alert on any execution errors, then this requirement is satisfied.

5.4.3. DDL- and table-readability

This requirement is evaluated through review of the code with the issues discussed in section 2.3.2 in mind (comments enhancing readability, unnecessary commands and names). The level of readability is not easy to establish, since it is a matter of opinion and different developers could have different views on it.

5.4.4. Clean data

The evaluation of this requirement is done through examining the DDL-code to see if stored procedures and triggers (if present) could be eliminated without losing any functionality.

6. Performing the quality tests

This chapter presents issues concerning the creation of the models, generation of DDL-code and results from executing the quality tests. Section 6.1 discuss issues concerning implementing the models in each tool. Section 6.2 presents issues concerning the generation of DDL-code in each tool. Section 6.3 presents the results from performing the tests.

6.1. Implementation of data models

This section presents issues concerning implementation of the models in each tool, created from the template models presented in section 5.3. The steps involved in creating the models and the resulting models are explained in appendix B1, B2, B3, C1, C2, C3, D1 and D2. Section 6.1.1 describes issues concerning modeling in PyDBDesigner, while section 6.1.2 describe issues concerning modeling in Rational Rose. Section 6.1.3 presents issues concerning modeling in ArgoUML; although this tool was not used in the examination of DDL-code, one of the template models (model 1) was implemented in it (and an attempt to generate DDL-code was made).

6.1.1. PyDBDesigner

Several issues arose while modeling with this tool and some special solutions had to be implemented. The issues were:

- Inheritance could not be modeled directly.
- Multi-valued attributes could not be modeled directly.
- Multi-level inheritance (and “ordinary” inheritance) is modeled through weak entities. Problems arise when a weak entity is “dependant” on another weak entity (such as the “Freq_trav” table in template data-model 1 in appendix A1). This problem is related to the model deficiencies described below.
- The relationship names appear to be static (not changeable), which could obscure the model.
- There is no “Date” data type.
- No way to put cardinality on the relations (this is done through putting the foreign keys in the “right” tables).
- The keys are not directly visible in the model, which could obscure the model.

Some of the above shortcoming made it impossible to model some parts of the template data models correctly (explained in the next section). The models are presented in appendix B2 and B3.

Model deficiencies

The model deficiencies are all related to relationships among tables, these are created through foreign keys (i.e. they cannot be created directly). The situations where PyDBDesigner behaved incorrectly are related to tables being referenced by one or several other tables.

As explained in appendix B1, every attribute in the entire model is added in a list (the “property list”) and relations are created when these attributes are “used” in more than one table. Unwanted relations were created when using the same attribute

6 Performing the quality tests

(referencing the same attribute in the property list) as foreign key in more than one table (i.e. when trying to create more than one relation to the same table), that is more than two tables are using the same attribute.

These deficiencies make the models in appendix B2 and B3 incomplete and give the proprietary tool an unplanned advantage over PyDBDesigner. To avoid errors in the DDL-code, the affected relations were deleted.

6.1.2. Rational Rose

The following structures cannot be modeled directly in Rose and had to be modeled using alternative strategies:

- Inheritance, modeled using a composite relation (as proposed by the documentation).
- “many-to-many” (m:n) relations, modeled with an extra table.
- Multi-valued attributes (such as “friends” in the Client table, in model 1), modeled with an extra table.

Despite these shortcomings in Rose, it appears more appropriate for data modeling, than PyDBDesigner (which was expected, since Rose is considered much more mature than PyDBDesigner).

6.1.3. ArgoUML

Issues concerning the modeling activity in ArgoUML are:

- Keys are modeled through tagged values (hidden type-value pairs), which are not seen directly in the model (which in turn obscures the model).
- “many-to-many” (m:n) relations can not be modeled directly (modeled with an extra table).
- Multi-valued attributes cannot be modeled directly (modeled with an extra table).

Despite these shortcomings ArgoUML was successfully used to create the entire model (only model 1 was implemented). One advantage that it had over PyDBDesigner and Rose is that inheritance could be modeled directly (which makes the model clearer).

6.2. Generation of DDL-files

This section explains issues concerning the generation of DDL-code from each tool; the complete set of steps involved in generating the DDL-code is explained in appendix B1, C1 and D1, appendix B4, B5, C4 and C5 contains the DDL-files (generated from PyDBDesigner and Rational Rose). Section 6.3.1 describes issues concerning DDL-generation in PyDBDesigner, while section 6.3.2 describe issues concerning DDL-generation in Rational Rose. Section 6.3.3 describe the attempt made to generate DDL-code from a model created in ArgoUML.

6.2.1. PyDBDesigner

The generation of DDL-code in PyDBDesigner was easy and no problems arose while generating the code. The steps involved in the generation are explained in appendix B1.

6.2.2. Rational Rose

The generation of DDL-code in Rational Rose was also easy and no problems arose while generating the code. The steps involved in the generation are explained in appendix C1.

6.2.3. ArgoUML

The generation tool (uml2sql) is immature and the use could be problematic for developers with little or no experience with this type of OSS and use of Java (such as the researcher). The model needs to follow a set of rules (as explained in appendix D1), which are explained with little detail in the tool documentation; this also complicated the procedure.

The DDL-generation procedure (from the ArgoUML-model) was problematic from the start (installation of uml2sql). Uml2sql is (as previously mentioned) immature; some aspects of the tool also indicate this, such as:

- No support for stored procedures (modeled procedures are ignored).
- The GUI (graphical user interface)-version does not seem to work at all.

The GUI-version of the tool gave the following error message when attempting to execute it:

Main method not public.

After editing the source code, changing the main method to public (line 572 in DBSyncGUI.java) and compiling, the following error message was displayed when attempting to execute:

Failed to load Main-Class manifest attribute from uml2sql.jar

Since the researcher is not an experienced programmer the attempt to fix the GUI-version ended here and the text-based version was used instead.

The text-based version did not work either (but unlike the GUI-version, it could be executed); the error messages concerned the XML-code of the model-file (to many errors were displayed to list here). The model was checked, but no errors could be found (the procedure was also attempted with a simple model, with only one table) and it was concluded that the tool is not working properly or the researcher had severely (since the number of error messages was large) misinterpreted the instructions in appendix D1.

6.3. Results from performing the quality tests

This section presents the results from executing the tests presented in section 5.4. As explained in section 6.2.3 DDL-generation using ArgoUML was not possible and is therefore not explained in this section.

6.3.1. DDL-consistency

To satisfy this requirement the DDL-code must describe the same database as the model.

Information preservation

One way to further verify the DDL-code is to use reverse engineering (i.e. generate a model from DDL-code), this is only possible in Rose and not in PyDBDesigner. The procedure was however performed with two DDL-files (only DDL-file 1 from both

6 Performing the quality tests

tools) and the results (and steps involved in performing it) are presented in appendix E.

PyDBDesigner:

All the attributes present in the model (model created in the tool) are also present in the DDL-code, with the right type.

The relations in this tool are, as mentioned earlier, created through foreign keys (not directly modeled); all the foreign keys created with the tool are also implemented in the DDL-code.

Inheritance could not be modeled with this tool.

Reverse engineering in Rose, using this DDL-file was unsuccessful; this is due to faulty DDL-code (section 6.3.2).

Rose:

This tool also generated the attributes correctly.

The relations in Rose can be modeled directly (unlike PyDBDesigner) and the tool adds foreign keys automatically.

Inheritance could not be modeled directly, but instead through the use of composite relations (section 6.1.2); this is correctly implemented in the DDL-code.

The resulting model from reverse engineering Rose DDL-code 1 appears to describe the same database as the original model.

Implicit integrity constraints

PyDBDesigner:

No constraints were added in the DDL-file.

Rose:

Integrity constraints are added for each foreign key, the actions can be specified in the model. The default value is “no action”, for each type of event.

Inheritance

Inheritance could not be modeled directly (which is, as explained in section 2.3.2, required to evaluate this aspect) with any of the tools.

6.3.2. DDL-functionality

To satisfy this requirement the DDL-code must be executable on the DBMS (section 5.2.3).

PyDBDesigner:

The tool failed this test (with both DDL-files); it would however not fail it if the model did not contain any empty tables (tables without any attributes). The model was not meant to contain any empty tables; these were there since it was not possible to model the entire model correctly (see section 6.1.1).

Rose:

The Rose DDL-files could be executed on the DBMS without any errors being displayed.

6.3.3. DDL- and table-readability

The readability of both DDL-files is not optimal, and some information is even lost in the DDL-file, such as relationship names (only available in the rose model). This can

6 Performing the quality tests

be seen from the rose model in appendix E (created through reverse engineering); the names could have been saved through the use of some type of comment.

Comments enhancing readability

No comments are generated by any of the tools.

Unnecessary commands

PyDBDesigner:

The primary keys in each table were added through “alter table”-commands after each table definition, which could instead be written in the table definition. The corresponding “alter table”-commands for foreign keys were generated at the end of the DDL-file, these could also have been written in the table definition.

Rose:

All primary and foreign keys are modeled through the use of constraints (which are named). The primary key constraints are added in the table definition, while the foreign key constraints are added at the end of the DDL-file. Both the primary and foreign keys could have been written in the table definition.

Names

The names of tables and attributes are consistent with the model, in both tools.

6.3.4. Clean data

The DDL-code should not contain any unnecessary amount of triggers and stored procedures.

PyDBDesigner:

Triggers and stored procedures could not be created with this tool (and none were generated automatically).

Rose:

No stored procedures are present in the DDL-code (only created if modeled) and no triggers are created (also only created if modeled).

7. Analysis of results

This chapter contains an analysis of the results of this study. Section 7.1 contains an analysis. Section 7.2 contains a comparison of the results with related research.

7.1. Analysis

The results in chapter 6 clearly indicate that the proprietary tool is superior to the open source tool. The results of performing the tests presented in chapter 5 on the open source tool compared to the results of the tests performed on the proprietary tool are summarized in Table 9.

Requirement	Result
1. DDL-consistency	The result is acceptable, the model was however not complete, which made the proprietary tool superior.
2. DDL-functionality	The proprietary tool passed this test with no problems, while the DDL-files generated by the open source tool failed to execute (unless manually edited).
3. DDL- and table-readability	The proprietary tool seems to perform better with respect to this requirement (this is however, as previously mentioned, a matter of opinion).
4. Clean data	The results of evaluating this requirement on both tools are the same.

Table 9: Summary of results

One result that might draw special attention is number 2, where DDL-functionality is evaluated. The execution of the DDL-files generated from PyDBDesigner was a failure, since empty tables (tables with no attributes) exists; which is not accepted by the DBMS. A test with Rose was performed, where empty tables were modeled, to see if it behaved the same way. Rose did not generate these tables (even if they existed in the model). After manual removal of these tables (e.g. Friends, Passenger, and Agency in model 1), the DDL-file was executable.

The overall results suggest that PyDBDesigner does not perform much worse than Rose. It would however be misleading to only consider them; instead the issues discussed in section 6.1.1 (modeling deficiencies) should also be taken into consideration, which is likely to change the view of the tool. With these issues in mind, Rational Rose seems much superior to PyDBDesigner.

7.2. Comparison with related research

There have been a number of studies dealing with open source, such as Stamelos et al. (2002), McConnell (1999), Feller and Fitzgerald (2000), Robbins and Redmiles (2000). These are related to this study in that they study the quality of open source, even though most of them do not study open source CASE. The results of this study are similar to the results of these studies.

7 Analysis of results

Stamelos et al. (2002) perform a study dealing with source code quality of open source software. They conclude that the quality does not reach industrial standard, but is better than opponents of open source might expect. The results of this study partly concur with these findings; this study does however indicate that open source CASE-tools is not ready for extensive use.

This study is focused on DDL-code generated by the tools, there has however also been some evaluation of the modeling capabilities of the tools. Robbins and Redmiles (2000) evaluate ArgoUML in their study, they encourage the use of this tool. The results of the evaluation (although not complete) of ArgoUML in this study do not disagree with these findings, since most modeling structures could be created (some, such as inheritance, even better (clearer) than Rose) with ArgoUML.

All of the mentioned research indicates that open source software is not ready for industrial use. The results of this study concur with these conclusions, they do however also indicate that open source is not ready for extensive use.

8. Conclusions

8.1. Summary

This section provides a summary of the results with regard to the aim of this study.

Aim of this study

To examine quality of DDL-code generated by open source database modeling CASE-tools and compare the results with a similar examination performed on a proprietary database modeling CASE-tool.

The aim of this study has been fulfilled through doing a case study examining two (three) representative tools from each type of tool and comparing these with each other. The study indicates that open source CASE-tools today are not ready to challenge proprietary tools, when it comes to database modeling and DDL-generation.

Open source software are developed very fast compared to proprietary software (as previously mentioned) and could produce a more capable modeling tool in the near future.

8.2. Contributions

This is a single case study examining a small number of CASE-tools that are available; the results of examining the proprietary tool cannot and is not meant to be generalized to all proprietary tools (used with database modeling), they are only used as comparison to the results of examining an open source tool. The main contributions of this study lie in the examination of the open source tools.

The study indicate that open source CASE-tools are not ready for extensive use. The obvious success of some open source projects (such as Linux and Apache) should however not be forgotten, this study only deals with a small subset of open source and is not meant to be generalized over all types of open source software.

The quality tests were performed using two complete models, with the same results. This suggests that the results could be generalized over most models, since the deficiencies are similar in both models (for example the modeling deficiencies are closely related in both models).

8.3. Discussion

This section contains a discussion of the results, focusing on their relevance and the research method used to obtain them.

8.3.1. Relevance of the results

Even though this is a single case study, it seems to be possible to generalize the results over all open source CASE-tools used for database modeling (with respect to DDL-generation) that exist today, since the number of these tools is small and all tools that were found where tested before choosing tools for this study (which indicated that the chosen tools were most mature). The use of two large models to test the tools, with similar results, also supports this.

The relevance of the results is subject to some discussion, since the tested tools (excluding ArgoUML) are at an early stage of their development and there does not exist many more similar open source tools. The results are relevant in that they could

influence a decision of using open source. However, as mentioned earlier, open source software tends to evolve rapidly and the significance of the results from this study could decrease in a near future, since the quality of the tools is likely to change.

8.3.2. Research method

Considering the aim of this study, the research method seems to be adequately successful. This is much due to the small number of open source CASE-tools that are available. If this was not the case, the study would likely need more planning and a single case study would in that case produce less significant results, since large variations are likely to exist among open source tools.

8.3.3. Problems with the tools

ArgoUML was the open source tool originally chosen for this study, but it was not chosen since the DDL-generation tool (uml2sql) does not seem to be working. This probably influenced the results, since ArgoUML seems more mature than PyDBDesigner and might have performed better. ArgoUML is however likely to evolve and might include DDL-generation capabilities in the near future; with this feature and the already useful modeling capabilities, it could become a worthy opponent to proprietary tools. It is also worth mentioning that ArgoUML has evolved into Poseidon for UML⁹ (Gentleware), a proprietary CASE-tool. Poseidon has, according to the vendor, over 300 000 users.

8.4. Possible future work

Open source is still a relatively unexplored area, especially when it comes to CASE-tools. There exist a lot of possible research, which could be aimed at many different aspects of open source CASE-tools (those built for database modeling and others).

One possible study could be focused on the modeling aspect of a tool, for example the level of clarity (e.g. how much “work-arounds” is necessary) and adherence to standards (such as UML and ER).

Another possible topic could deal with reverse engineering using an open source tool; this is an extensive area and there exist several types of reverse engineering (such as DDL-code to model and source code from different programming languages to a model).

The study could also deal with quality of other types of open source tools, such as operating systems or web-browsers. A comparison to similar proprietary tools could also be included in the study.

⁹ <http://www.gentleware.com/>

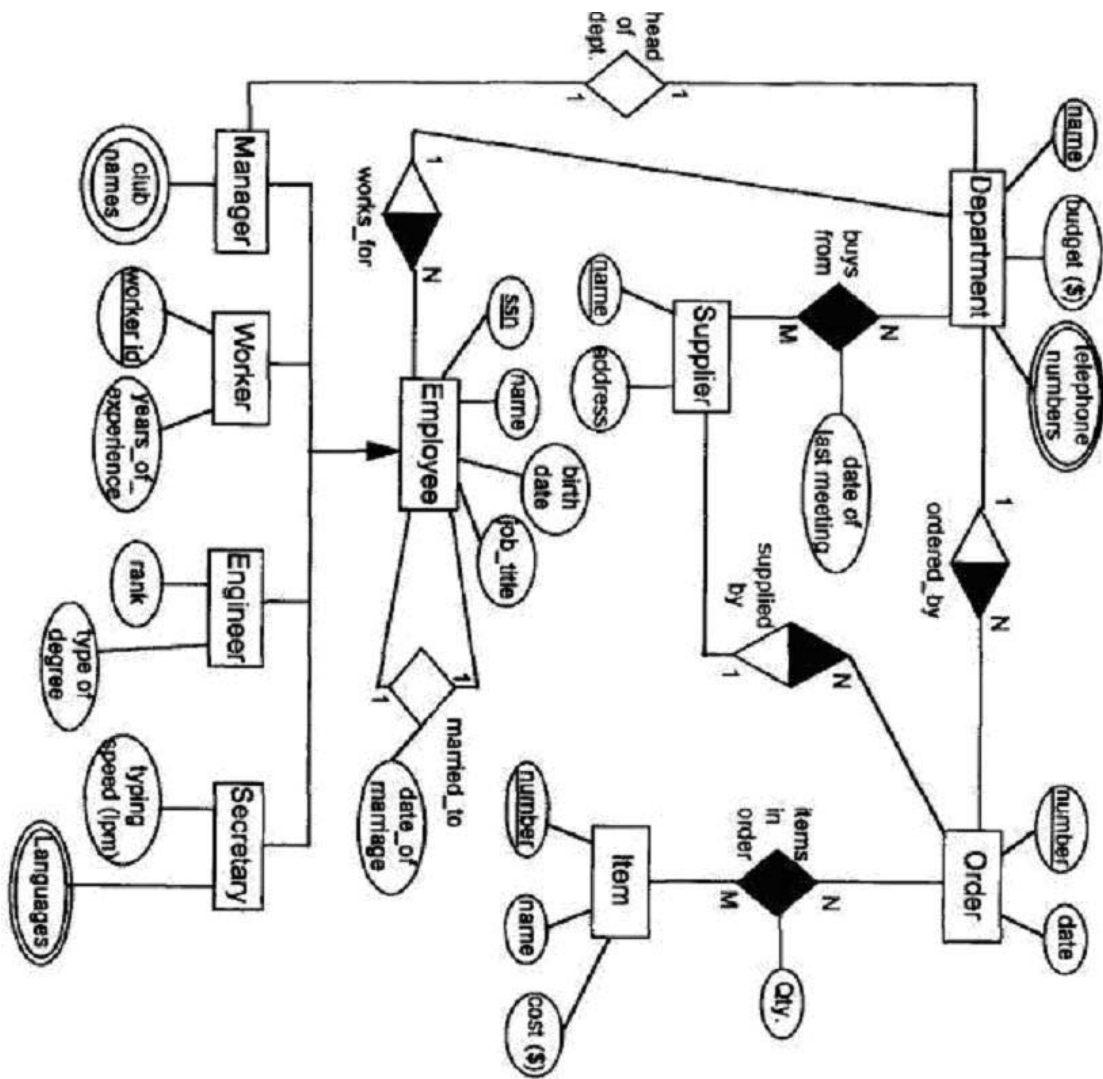
References

- Benbasat, I., Goldstein, D.K., and Mead, M. (1987) The Case Research Strategy in Studies of Information Systems, *MIS Quarterly*, **11** (3), 369-386.
- Berndtsson, M., Hansson, J., Olsson, B., and Lundell, B. (2002) *Planning and implementing your final year project with success!* Springer.
- Chen, P. (1976) The Entity-Relationship model: towards a unified view of data, *ACM Transactions on database systems*, **1** (1), 9-36.
- Elmasri, R., and Navathe S. (2000) *Fundamentals of database systems third edition*. Addison-Wesley.
- Fahrner, C., and Vossen, G. (1995) A survey of database design transformations based on the Entity-Relationship model, *Data & Knowledge Engineering*, **15** (3), 213-250.
- Feller, J., and Fitzgerald, B. (2000) A Framework Analysis of the Open Source Software Development Paradigm, *Proceedings of the 21st Annual International Conference on Information Systems*.
- Fowler, M., and Scott, K. (1999) *UML Distilled second edition-a brief guide to the standard object modeling language*. Addison-Wesley.
- Gallivan, M. J. (2001) Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies, *Information Systems Journal*, **11** (4), 277-304.
- Hansen, M., Köhntopp, K., and Pfitzmann, A. (2002) The Open source approach – opportunities and limitations with respect to security and privacy, *Computers & security*, **21** (5), 461-471.
- Hoxmeier, J. A. (1998) Typology of database quality factors, *Software quality journal*, **7** (3), 179-193.
- Jankowski, D. (1997) Computer-aided software systems, *Empirical Software Engineering: an international Journal*, **2**, 11-38.
- King, S. F. (1996) CASE tools and organizational action, *Information Systems Journal*, **6** (3), 173-194.
- King, S. (1997) Tool support for systems emergence: A multimedia CASE tool, *Information and software technology*, **39** (5), 323-330.
- Kolp, M., and Zimányi, E. (2000) Enhanced ER to relational mapping and interrelational normalization, *Information and software technology*, **42**, 1057-1073.
- Lee, A. S. (1989) A Scientific Methodology for MIS Case Studies, *MIS Quarterly*, **13** (1), 33-52.
- Levitin, A., and Redman, T. (1995) Quality dimensions of a conceptual view, *Information Processing & Management*, **31** (1), 81-88.
- Maansaari, J., and Iivari, J. (1999) The evolution of CASE usage in Finland between 1993 and 1996, *Information and Software Technology*, **36** (1), 37-53.
- Maxwell, J. A. (1996) *Qualitative research design: an interactive approach*. Sage publications.

References

- McConnell, S. (1999) Open source methodology: ready for prime time?, *IEEE Software*, **16** (4), 6-8.
- O'Reilly, T. (1999) Lessons from open-source software development, *Communications of the ACM*, **42** (4), 33-37.
- Perens, B. (1997) "The Open source Definition"
(http://www.opensource.org/docs/definition_plain.php; accessed February, 2003).
- Piattini, M., Calero, C., and Genero, M. (2001) Table oriented metrics for relational databases, *Software quality journal*, **9** (2), 79-97.
- Post, G. and Kagan, A. (2000) OO-CASE tools: an evaluation of Rose, *Information and Software Technology*, **42** (6), 383-388.
- Pressman, R. (2000) *Software engineering, a practioner's approach* (5:th edition). Mc Graw Hill.
- Raymond, E. S. (1998) "The cathedral and the bazaar"
(http://www.firstmonday.dk/issues/issue3_3/raymond/; accessed February, 2003).
- Rehbinder, A. (2000) On Applying a Method for Developing Context Dependent CASE-tool Evaluation Frameworks, HS-IDA-MD-00-012, Master's Dissertation, Department of Computer Science, University of Skövde, Skövde, Sweden.
- Robbins, J. E., and Redmiles, D. F. (2000) Cognitive support, UML adherence, and XMI interchange in Argo/UML, *Information and Software Technology*, **42** (2), 79-89.
- Shoval, P., and Shiran, S. (1997) Entity-relationship and object-oriented data modeling - an experimental comparison of design quality, *Data & Knowledge Engineering*, **21**, 197-315.
- Stamelos, I., Angelis, L., and Apostolos, O. (2002) Code quality analysis in open source software development, *Information Systems Journal*, **12** (1), 43-60.
- Yin, R. K. (1994) *Case study research: Design and methods* (2nd edition). Thousand Oaks.

Appendix A2: Template data model 2



Template data model 2 (Shoval and Shiran, 1997, p. 310)

Appendix B1: Steps involved in creating model and generating DDL-code in PyDBDesigner

This section describes the steps involved in creating the model and generating the DDL-script; they are very specific to PyDBDesigner (version 0.1.3). The tool is under heavy development (according to the creators) and some parts of the model could not be created (explained further in section 6.1.1). It should also be noted that the steps described below is one of a number of ways to accomplish the task.

Creating the model

Step 0

To be able to execute PyDBDesigner the following software has to be downloaded and installed:

Python (<http://www.python.org>; accessed May, 2003): The Python interpreter (version 2.2.2 used in this study).

WxPython (<http://www.wxpython.org>; accessed May, 2003): A GUI toolkit for Python (version 2.4.0 used in this study)

It could get tiresome to use the menus (there is no toolbar available) and it might be more convenient (and efficient) to use the shortcut keys, the ones used are described in Table 10.

Action	Shortcut	Menu option
Edit the property list	Ctrl+P	Tools - Edit property list
Create table	Ctrl+E	Tools - Add entity
Edit table	Ctrl+U	Tools - Edit entity
Save model	Ctrl+S	File - Save

Table 10: Shortcut options

Step 1

Every property in the entire model has to be created through adding it to the "property list", this could be done before creating each table or all the properties could be added at one time. To distinguish between the properties (in this model), each property-name is preceded with the table-name that the property belongs to, for example "Client_Name" (the name attribute in the Client table). The property list is accessed through Ctrl+P, adding each property consists of pressing the "Add" button and filling in the name, type and size of the new property. After all properties have been added, the "OK" button is pressed.

Step 2

This step consists of adding each table and editing the tables to change name and add properties. A table is added by pressing Ctrl+E and edited by pressing Ctrl+U. In the "Edit entity"-window the name is changed and properties are added. A property is added by pressing the Add-button and choosing the property from the property list (created in step 1) and then choosing a "local name" (the name that the property will

B1 Steps involved in creating model and generating DDL-code in PyDBDesigner

have in that table) and choosing if the property is primary, Unique, Not null and/or editable. When all properties are added (and the name is changed), the OK-button is pressed.

Step 3

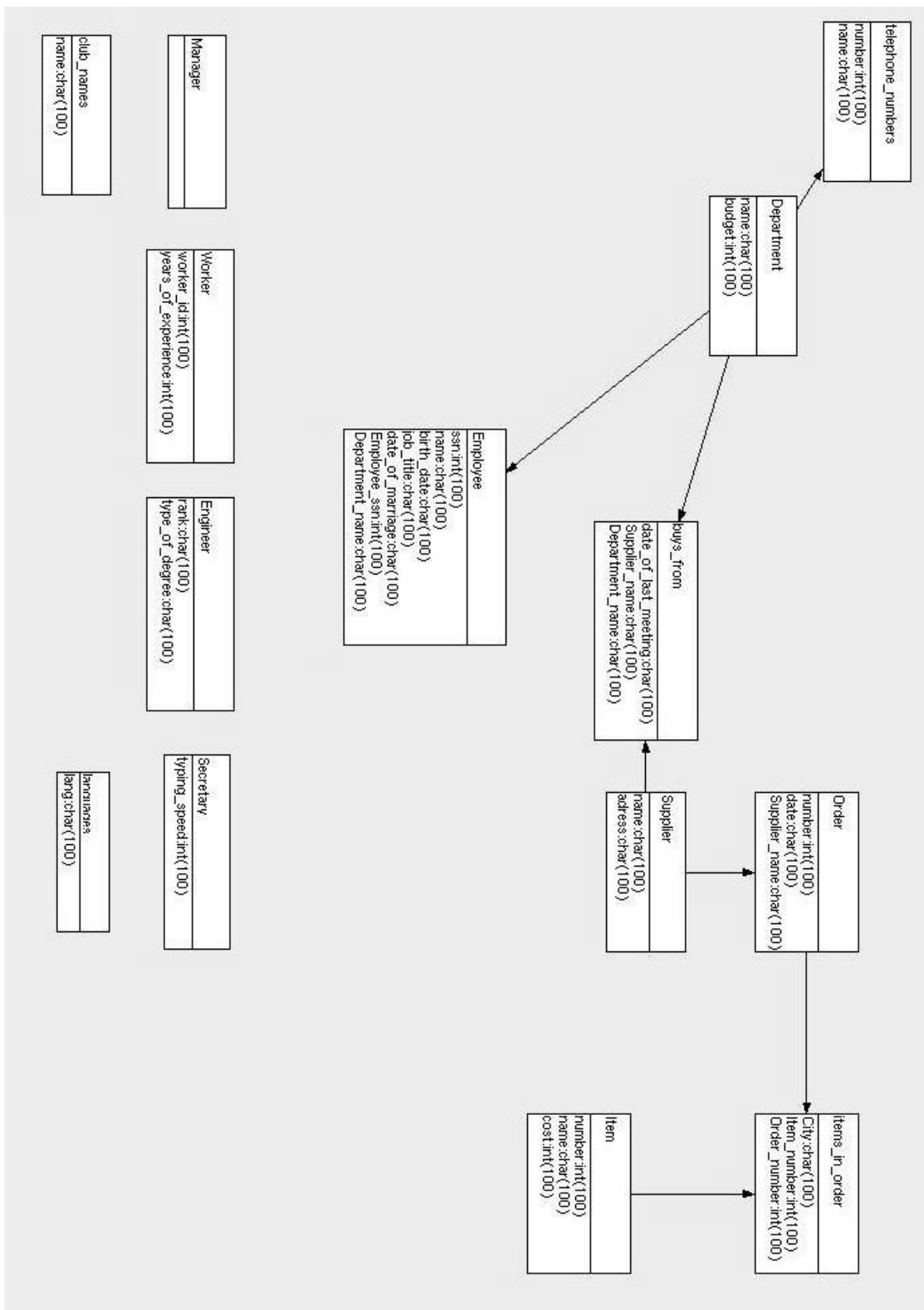
This step consists of generating the relations. Relations are created through the use of foreign keys; these are added through adding the same property to two tables (but not necessarily the same local name). See previous step for instructions on how to add a property. After adding the foreign keys, the relations have to be “generated”; this is done through accessing “Tools”-“Generate all relations”. This can be done either after adding each foreign key (recommended, since errors will be detected earlier) or after adding all the foreign keys.

Generating the DDL-file

This step is relatively straightforward and consists of the following steps:

1. Accessing the “Save as SQL“ option from the “File”-menu.
2. Choosing path and entering the filename of the DDL-file.

Appendix B3: PyDBDesigner data model 2



Appendix B4: PyDBDesigner DDL-file 1

```

create table Client(
    Name char(100) not null,
    Telephone int(50) not null
);
alter table Client add primary key (Name, Telephone);
create table hold_reser(
    seat_num int(50),
    hosmoking_option char(10),
    Trip_number int(50) not null,
    Segment_number int(50) not null,
    Date char(50) not null,
    Name char(100) not null,
    Telephone int(50) not null
);
alter table hold_reser add primary key (Trip_number, Segment_number,
Date, Name, Telephone);
create table Dail_R_Seg(
    Trip_number int(50) not null,
    Segment_number int(50) not null,
    Date char(50) not null,
    reserved_seats int(50),
    Dail_R_Seg_available_seats int(50),
    Rte_segm_Trip_number int(50),
    Rte_segm_Segment_number int(50),
    Dail_trip_Trip_number int(50),
    Dail_trip_Date char(50)
);
alter table Dail_R_Seg add primary key (Trip_number, Segment_number,
Date);
create table Freq_trav(
    Freq_trav_Mileage int(50)
);
create table Friends(
);
create table Passenger(
);
create table Agency(
);
create table is_on_board(
    Trip_number int(50) not null,
    Segment_number int(50) not null,
    Date char(50) not null,
    Name char(100) not null,
    Telephone int(50) not null
);
alter table is_on_board add primary key (Trip_number, Segment_number,
Date, Name, Telephone);
create table Rte_segm(
    Trip_number int(50) not null,
    Segment_number int(50) not null,
    depart_time char(50),
    arriv_time char(50),
    distance int(50),
    price int(50),
    Trip_Trip_number int(50)
);
alter table Rte_segm add primary key (Trip_number, Segment_number);
create table Bus(
    License_number char(10) not null,
    bus_id char(10) not null,

```

B4 PyDBDesigner DDL-file 1

```
        seats int(50),
        make char(10),
        last_check char(50)
    );
alter table Bus add primary key (License_number, bus_id);
create table Dail_trip(
    Trip_number int(50) not null,
    Date char(50) not null,
    Duration int(50),
    License_number char(10),
    bus_id char(10),
    Driver_id int(50),
    Trip_Trip_number int(50)
);
alter table Dail_trip add primary key (Trip_number, Date);
create table Trip(
    Trip_number int(50) not null,
    Weekdays char(10),
    number_of_segment int(50),
    dep_city char(100),
    arr_city char(100)
);
alter table Trip add primary key (Trip_number);
create table Ordinary(
);
create table Special(
    event char(100)
);
create table Bus_prob(
    Problem_number int(50),
    Description char(200),
    License_number char(10),
    bus_id char(10)
);
create table Driver(
    Driver_id int(50) not null,
    Name char(100),
    record char(100),
    adress char(100),
    license_type char(10),
    Driver_Driver_id int(50)
);
alter table Driver add primary key (Driver_id);
create table Driver_abs(
    Date char(50) not null,
    Cause char(100),
    Driver_id int(50) not null
);
alter table Driver_abs add primary key (Date, Driver_id);
alter table Driver_abs add foreign key (Driver_id) references Driver
(Driver_id);
alter table Driver add foreign key (Driver_id) references Driver
(Driver_id);
alter table Bus_prob add foreign key (License_number, bus_id)
references Bus (License_number, bus_id);
alter table Dail_trip add foreign key (License_number, bus_id)
references Bus (License_number, bus_id);
alter table Dail_trip add foreign key (Driver_id) references Driver
(Driver_id);
alter table Dail_trip add foreign key (Trip_Trip_number) references
Trip (Trip_number);
```

B4 PyDBDesigner DDL-file 1

```
alter table Rte_segm add foreign key (Trip_Trip_number) references
Trip (Trip_number);
alter table Dail_R_Seg add foreign key (Rte_segm_Trip_number,
Rte_segm_Segment_number) references Rte_segm (Trip_number,
Segment_number);
alter table Dail_R_Seg add foreign key (Dail_trip_Trip_number,
Dail_trip_Date) references Dail_trip (Trip_number, Date);
alter table hold_reser add foreign key (Trip_number, Segment_number,
Date) references Dail_R_Seg (Trip_number, Segment_number, Date);
alter table is_on_board add foreign key (Trip_number,
Segment_number, Date) references Dail_R_Seg (Trip_number,
Segment_number, Date);
alter table hold_reser add foreign key (Name, Telephone) references
Client (Name, Telephone);
alter table is_on_board add foreign key (Name, Telephone) references
Client (Name, Telephone);
alter table hold_reser add foreign key (Trip_number, Segment_number,
Date) references is_on_board (Trip_number, Segment_number, Date);
```

Appendix B5: PyDBDesigner DDL-file 2

```
create table club_names(  
    name char(100)  
);  
create table Manager(  
);  
create table Worker(  
    worker_id int(100) not null,  
    years_of_experience int(100)  
);  
alter table Worker add primary key (worker_id);  
create table Engineer(  
    rank char(100),  
    type_of_degree char(100)  
);  
create table Secretary(  
    typing_speed int(100)  
);  
create table languages(  
    lang char(100)  
);  
create table Employee(  
    ssn int(100) not null,  
    name char(100),  
    birth_date char(100),  
    job_title char(100),  
    date_of_marriage char(100),  
    Employee_ssn int(100),  
    Department_name char(100)  
);  
alter table Employee add primary key (ssn);  
create table telephone_numbers(  
    number int(100),  
    name char(100)  
);  
create table Department(  
    name char(100) not null,  
    budget int(100)  
);  
alter table Department add primary key (name);
```

B5 PyDBDesigner DDL-file 2

```
create table buys_from(
    date_of_last_meeting char(100),
    Supplier_name char(100) not null,
    Department_name char(100) not null
);
alter table buys_from add primary key (Supplier_name,
Department_name);
create table Supplier(
    name char(100) not null,
    adress char(100)
);
alter table Supplier add primary key (name);
create table Order(
    number int(100) not null,
    date char(100),
    Supplier_name char(100)
);
alter table Order add primary key (number);
create table items_in_order(
    City char(100),
    Item_number int(100) not null,
    Order_number int(100) not null
);
alter table items_in_order add primary key (Item_number,
Order_number);
create table Item(
    number int(100) not null,
    name char(100),
    cost int(100)
);
alter table Item add primary key (number);
alter table Employee add foreign key (ssn) references Employee
(ssn);
alter table Employee add foreign key (Department_name) references
Department (name);
alter table items_in_order add foreign key (Order_number) references
Order (number);
alter table items_in_order add foreign key (Item_number) references
Item (number);
alter table buys_from add foreign key (Department_name) references
Department (name);
alter table buys_from add foreign key (Supplier_name) references
Supplier (name);
```

B5 PyDBDesigner DDL-file 2

```
alter table Order add foreign key (Supplier_name) references  
Supplier (name);  
alter table telephone_numbers add foreign key (name) references  
Department (name);
```

Appendix C1: Steps involved in creating model and generating DDL-code in Rational Rose

This section describes the steps involved in creating the model and generating the DDL-script; they are very specific to Rational Rose and might also be specific to this version (2002.05.00). It should also be noted that the steps described below is one of a number of ways to accomplish the task.

Creating the model

Step 1

The first step involves creating a schema. This is done by right clicking on the “logical view” in the tree-view (usually placed to the left) and choosing “schema” from the “data modeler”-“new” menu.

Step 2

After performing step 1, a schema symbol appears in the tree-view. When creating new tables, relations, and procedures etc. the schema symbol is right-clicked and a structure is chosen from “data modeler”-“new” menu. The created components are edited (setting name, adding attributes, setting primary key etc.) by double-clicking on their name in the tree-view.

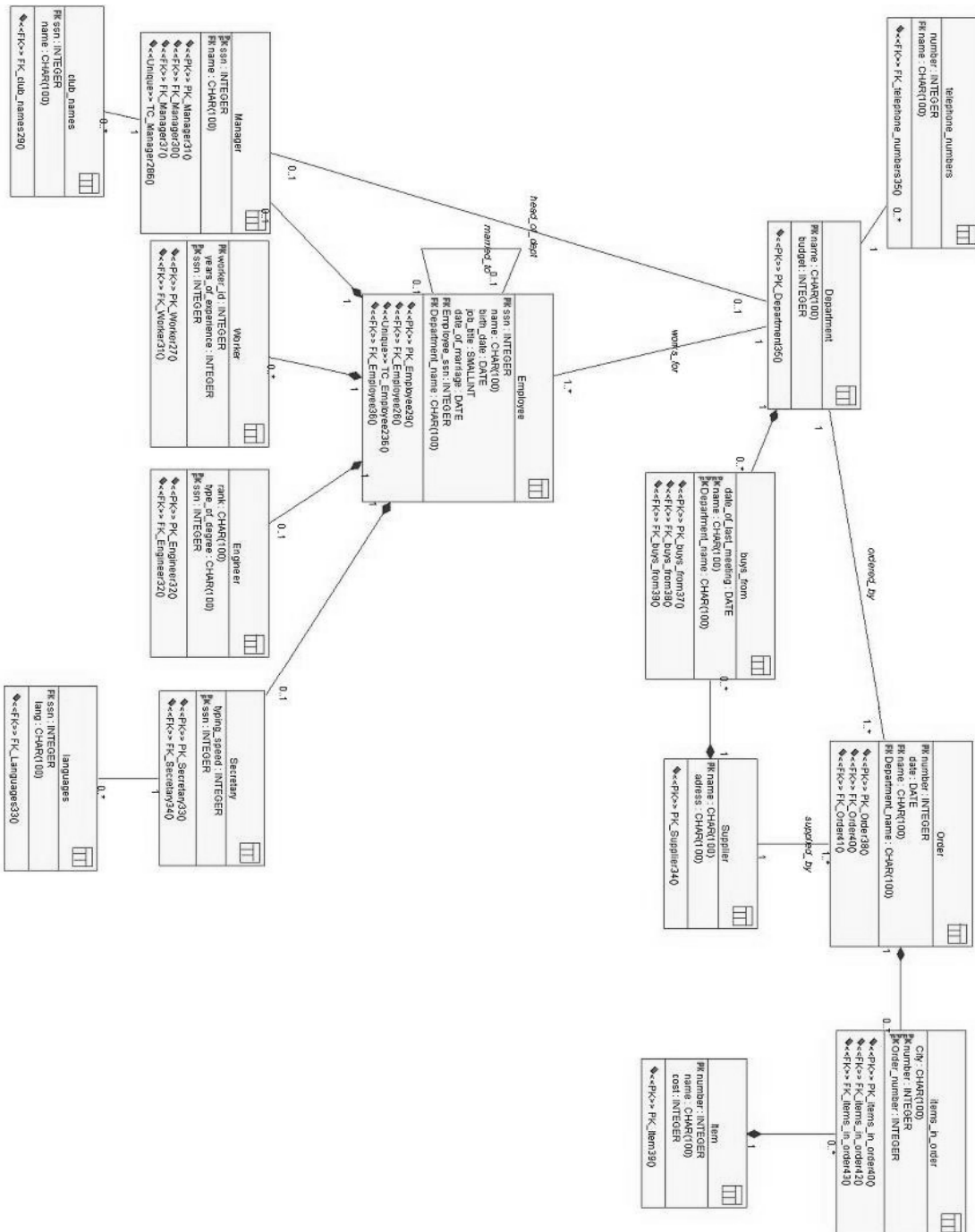
Step 3

After creating all the components of the model a diagram is created separately in order to visualize the model. This is done by choosing the “data model diagram”-option in the “new” menu in “data modeler” (also done by right-clicking on the schema symbol).

Generating the DDL-file

The DDL-generation is relatively easy in Rational Rose. The schema icon on left is right-clicked and “Forward engineer...” is chosen from the “Data modeler”-menu; which causes a “wizard” to appear. After choosing what to generate (triggers, index etc.) and file location, the generation process is complete.

Appendix C3: Rational Rose data model 2



Appendix C4: Rational Rose DDL-file 1

```

CREATE TABLE is_on_board (
    Name CHAR ( 100 ) NOT NULL,
    Telephone INTEGER NOT NULL,
    Trip_number INTEGER NOT NULL,
    Segment_number INTEGER NOT NULL,
    Date DATE NOT NULL,
    CONSTRAINT PK_is_on_board6 PRIMARY KEY (Trip_number,
Segment_number, Date, Name, Telephone)
);

CREATE TABLE Rte_segm (
    Trip_number INTEGER NOT NULL,
    Segment_number INTEGER NOT NULL,
    depart_time TIME,
    arriv_time TIME,
    distance INTEGER,
    price INTEGER,
    COL_48 INTEGER NOT NULL,
    CONSTRAINT PK_Rte_segml4 PRIMARY KEY (Trip_number,
Segment_number)
);

CREATE TABLE Client (
    Name CHAR ( 100 ) NOT NULL,
    Telephone INTEGER NOT NULL,
    CONSTRAINT PK_Client0 PRIMARY KEY (Name, Telephone)
);

CREATE TABLE Freq_trav (
    Mileage INTEGER,
    Name CHAR ( 100 ) NOT NULL,
    Telephone INTEGER NOT NULL,
    CONSTRAINT PK_Freq_trav4 PRIMARY KEY (Name, Telephone)
);

CREATE TABLE Agency (
    Name CHAR ( 100 ) NOT NULL,
    Telephone INTEGER NOT NULL,
    CONSTRAINT PK_Agency3 PRIMARY KEY (Name, Telephone)
);

CREATE TABLE Dail_R_Seg (
    Trip_number INTEGER NOT NULL,
    Segment_number INTEGER NOT NULL,
    Date DATE NOT NULL,

```

C4 Rational Rose DDL-file 1

```
reserved_seats INTEGER,
available_seats INTEGER,
COL_15 INTEGER NOT NULL,
COL_16 DATE NOT NULL,
COL_45 INTEGER NOT NULL,
COL_46 INTEGER NOT NULL,
CONSTRAINT PK_Dail_R_Seg7 PRIMARY KEY (Trip_number,
Segment_number, Date)
);

CREATE TABLE Dail_trip (
Trip_number INTEGER NOT NULL,
Date DATE NOT NULL,
Duration INTEGER,
bus_id CHAR ( 10 ) NOT NULL,
License_number CHAR ( 10 ) NOT NULL,
Driver_id INTEGER NOT NULL,
COL_49 INTEGER NOT NULL,
CONSTRAINT PK_Dail_trip8 PRIMARY KEY (Trip_number, Date)
);

CREATE TABLE Trip (
Trip_number INTEGER NOT NULL,
Weekdays CHAR ( 10 ),
number_of_segment INTEGER,
dep_city CHAR ( 100 ),
arr_city CHAR ( 100 ),
CONSTRAINT PK_Trip17 PRIMARY KEY (Trip_number)
);

CREATE TABLE Bus_prob (
Problem_number INTEGER,
Description CHAR ( 200 ),
bus_id CHAR ( 10 ) NOT NULL,
License_number CHAR ( 10 ) NOT NULL
);

CREATE TABLE Drivers_abs (
Date DATE NOT NULL,
Cause CHAR ( 100 ),
Driver_id INTEGER NOT NULL,
CONSTRAINT PK_Drivers_abs10 PRIMARY KEY (Driver_id, Date)
);

CREATE TABLE Driver (
Driver_id INTEGER NOT NULL,
```

C4 Rational Rose DDL-file 1

```
Name CHAR ( 100 ),
record CHAR ( 100 ),
adress CHAR ( 100 ),
license_type CHAR ( 10 ),
Driver_Driver_id INTEGER NOT NULL,
CONSTRAINT PK_Driver11 PRIMARY KEY (Driver_id)
);

CREATE TABLE hold_reser (
    seat_num INTEGER NOT NULL,
    smoking_option CHAR ( 10 ) NOT NULL,
    Name CHAR ( 100 ) NOT NULL,
    Telephone INTEGER NOT NULL,
    Trip_number INTEGER NOT NULL,
    Segment_number INTEGER NOT NULL,
    Date DATE NOT NULL,
    CONSTRAINT PK_hold_reser5 PRIMARY KEY (Trip_number,
Segment_number, Date, Name, Telephone)
);

CREATE TABLE Passenger (
    Name CHAR ( 100 ) NOT NULL,
    Telephone INTEGER NOT NULL,
    CONSTRAINT PK_Passenger2 PRIMARY KEY (Name, Telephone)
);

CREATE TABLE Ordinary (
    Trip_number INTEGER NOT NULL,
    CONSTRAINT PK_Ordinary18 PRIMARY KEY (Trip_number)
);

CREATE TABLE Bus (
    License_number CHAR ( 10 ) NOT NULL,
    bus_id CHAR ( 10 ) NOT NULL,
    seats INTEGER,
    make CHAR ( 10 ),
    last_check DATE,
    CONSTRAINT PK_Bus9 PRIMARY KEY (bus_id, License_number)
);

CREATE TABLE Friends (
    Name CHAR ( 100 ) NOT NULL,
    Telephone INTEGER NOT NULL
);

CREATE TABLE Special (
    event CHAR ( 100 ),
```

C4 Rational Rose DDL-file 1

```
Trip_number INTEGER NOT NULL,
CONSTRAINT PK_Special19 PRIMARY KEY (Trip_number)
);

ALTER TABLE is_on_board ADD CONSTRAINT FK_is_on_board4 FOREIGN KEY
(Name, Telephone) REFERENCES Client (Name, Telephone) ON DELETE NO
ACTION ON UPDATE NO ACTION;

ALTER TABLE is_on_board ADD CONSTRAINT FK_is_on_board5 FOREIGN KEY
(Trip_number, Segment_number, Date) REFERENCES Dail_R_Seg
(Trip_number, Segment_number, Date) ON DELETE NO ACTION ON UPDATE NO
ACTION;

ALTER TABLE Rte_segmn ADD CONSTRAINT FK_Rte_segmn18 FOREIGN KEY
(COL_48) REFERENCES Trip (Trip_number) ON DELETE NO ACTION ON UPDATE
NO ACTION;

ALTER TABLE Freq_trav ADD CONSTRAINT FK_Freq_trav2 FOREIGN KEY (Name,
Telephone) REFERENCES Passenger (Name, Telephone) ON DELETE NO
ACTION ON UPDATE NO ACTION;

ALTER TABLE Agency ADD CONSTRAINT FK_Agency1 FOREIGN KEY (Name,
Telephone) REFERENCES Client (Name, Telephone) ON DELETE NO ACTION
ON UPDATE NO ACTION;

ALTER TABLE Dail_R_Seg ADD CONSTRAINT FK_Dail_R_Seg8 FOREIGN KEY
(COL_15, COL_16) REFERENCES Dail_trip (Trip_number, Date) ON DELETE
NO ACTION ON UPDATE NO ACTION;

ALTER TABLE Dail_R_Seg ADD CONSTRAINT FK_Dail_R_Seg15 FOREIGN KEY
(COL_45, COL_46) REFERENCES Rte_segmn (Trip_number, Segment_number)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE Dail_trip ADD CONSTRAINT FK_Dail_trip19 FOREIGN KEY
(COL_49) REFERENCES Trip (Trip_number) ON DELETE NO ACTION ON UPDATE
NO ACTION;

ALTER TABLE Dail_trip ADD CONSTRAINT FK_Dail_trip10 FOREIGN KEY
(bus_id, License_number) REFERENCES Bus (bus_id, License_number) ON
DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE Dail_trip ADD CONSTRAINT FK_Dail_trip13 FOREIGN KEY
(Driver_id) REFERENCES Driver (Driver_id) ON DELETE NO ACTION ON
UPDATE NO ACTION;

ALTER TABLE Bus_prob ADD CONSTRAINT FK_Bus_prob11 FOREIGN KEY
(bus_id, License_number) REFERENCES Bus (bus_id, License_number) ON
DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE Drivers_abs ADD CONSTRAINT FK_Drivers_abs14 FOREIGN KEY
(Driver_id) REFERENCES Driver (Driver_id) ON DELETE NO ACTION ON
UPDATE NO ACTION;

ALTER TABLE Driver ADD CONSTRAINT FK_Driver24 FOREIGN KEY
(Driver_Driver_id) REFERENCES Driver (Driver_id) ON DELETE NO ACTION
ON UPDATE NO ACTION;

ALTER TABLE hold_reser ADD CONSTRAINT FK_hold_reser6 FOREIGN KEY
(Trip_number, Segment_number, Date) REFERENCES Dail_R_Seg
(Trip_number, Segment_number, Date) ON DELETE NO ACTION ON UPDATE NO
ACTION;

ALTER TABLE hold_reser ADD CONSTRAINT FK_hold_reser3 FOREIGN KEY
(Name, Telephone) REFERENCES Client (Name, Telephone) ON DELETE NO
ACTION ON UPDATE NO ACTION;
```

C4 Rational Rose DDL-file 1

```
ALTER TABLE Passenger ADD CONSTRAINT FK_Passenger0 FOREIGN KEY (Name,  
Telephone) REFERENCES Client (Name, Telephone) ON DELETE NO ACTION  
ON UPDATE NO ACTION;
```

```
ALTER TABLE Ordinary ADD CONSTRAINT FK_Ordinary20 FOREIGN KEY  
(Trip_number) REFERENCES Trip (Trip_number) ON DELETE NO ACTION ON  
UPDATE NO ACTION;
```

```
ALTER TABLE Friends ADD CONSTRAINT FK_Friends23 FOREIGN KEY (Name,  
Telephone) REFERENCES Client (Name, Telephone) ON DELETE NO ACTION  
ON UPDATE NO ACTION;
```

```
ALTER TABLE Special ADD CONSTRAINT FK_Special21 FOREIGN KEY  
(Trip_number) REFERENCES Trip (Trip_number) ON DELETE NO ACTION ON  
UPDATE NO ACTION;
```

Appendix C5: Rational Rose DDL-file 2

```

CREATE TABLE buys_from (
    date_of_last_meeting DATE NOT NULL,
    name CHAR ( 100 ) NOT NULL,
    Department_name CHAR ( 100 ) NOT NULL,
    CONSTRAINT PK_buys_from37 PRIMARY KEY (Department_name, name)
);

CREATE TABLE Worker (
    worker_id INTEGER NOT NULL,
    years_of_experience INTEGER,
    ssn INTEGER NOT NULL,
    CONSTRAINT PK_Worker27 PRIMARY KEY (ssn, worker_id)
);

CREATE TABLE items_in_order (
    City CHAR ( 100 ),
    number INTEGER NOT NULL,
    Order_number INTEGER NOT NULL,
    CONSTRAINT PK_items_in_order40 PRIMARY KEY (Order_number,
number)
);

CREATE TABLE club_names (
    ssn INTEGER NOT NULL,
    name CHAR ( 100 ) NOT NULL
);

CREATE TABLE Supplier (
    name CHAR ( 100 ) NOT NULL,
    adress CHAR ( 100 ),
    CONSTRAINT PK_Supplier34 PRIMARY KEY (name)
);

CREATE TABLE Manager (
    ssn INTEGER NOT NULL,
    name CHAR ( 100 ),
    CONSTRAINT TC_Manager286 UNIQUE (name),
    CONSTRAINT PK_Manager31 PRIMARY KEY (ssn)
);

CREATE TABLE Engineer (
    rank CHAR ( 100 ),
    type_of_degree CHAR ( 100 ),
    ssn INTEGER NOT NULL,
    CONSTRAINT PK_Engineer32 PRIMARY KEY (ssn)

```

C5 Rational Rose DDL-file 2

```
);  
CREATE TABLE Item (  
    number INTEGER NOT NULL,  
    name CHAR ( 100 ),  
    cost INTEGER,  
    CONSTRAINT PK_Item39 PRIMARY KEY (number)  
);  
CREATE TABLE languages (  
    ssn INTEGER NOT NULL,  
    lang CHAR ( 100 ) NOT NULL  
);  
CREATE TABLE Order (  
    number INTEGER NOT NULL,  
    date DATE,  
    name CHAR ( 100 ) NOT NULL,  
    Department_name CHAR ( 100 ) NOT NULL,  
    CONSTRAINT PK_Order38 PRIMARY KEY (number)  
);  
CREATE TABLE Employee (  
    ssn INTEGER NOT NULL,  
    name CHAR ( 100 ),  
    birth_date DATE,  
    job_title SMALLINT NOT NULL,  
    date_of_marriage DATE,  
    Employee_ssn INTEGER,  
    Department_name CHAR ( 100 ) NOT NULL,  
    CONSTRAINT TC_Employee236 UNIQUE (Employee_ssn),  
    CONSTRAINT PK_Employee29 PRIMARY KEY (ssn)  
);  
CREATE TABLE telephone_numbers (  
    number INTEGER NOT NULL,  
    name CHAR ( 100 ) NOT NULL  
);  
CREATE TABLE Secretary (  
    typing_speed INTEGER,  
    ssn INTEGER NOT NULL,  
    CONSTRAINT PK_Secretary33 PRIMARY KEY (ssn)  
);  
CREATE TABLE Department (  
    name CHAR ( 100 ) NOT NULL,  
    budget INTEGER,
```

C5 Rational Rose DDL-file 2

```
CONSTRAINT PK_Department35 PRIMARY KEY (name)
);

ALTER TABLE buys_from ADD CONSTRAINT FK_buys_from39 FOREIGN KEY
(Department_name) REFERENCES Department (name) ON DELETE NO ACTION
ON UPDATE NO ACTION;

ALTER TABLE buys_from ADD CONSTRAINT FK_buys_from38 FOREIGN KEY
(name) REFERENCES Supplier (name) ON DELETE NO ACTION ON UPDATE NO
ACTION;

ALTER TABLE Worker ADD CONSTRAINT FK_Worker31 FOREIGN KEY (ssn)
REFERENCES Employee (ssn) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE items_in_order ADD CONSTRAINT FK_items_in_order43 FOREIGN
KEY (Order_number) REFERENCES Order (number) ON DELETE NO ACTION ON
UPDATE NO ACTION;

ALTER TABLE items_in_order ADD CONSTRAINT FK_items_in_order42 FOREIGN
KEY (number) REFERENCES Item (number) ON DELETE NO ACTION ON UPDATE
NO ACTION;

ALTER TABLE club_names ADD CONSTRAINT FK_club_names29 FOREIGN KEY
(ssn) REFERENCES Manager (ssn) ON DELETE NO ACTION ON UPDATE NO
ACTION;

ALTER TABLE Manager ADD CONSTRAINT FK_Manager30 FOREIGN KEY (ssn)
REFERENCES Employee (ssn) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE Manager ADD CONSTRAINT FK_Manager37 FOREIGN KEY (name)
REFERENCES Department (name) ON DELETE NO ACTION ON UPDATE NO
ACTION;

ALTER TABLE Engineer ADD CONSTRAINT FK_Engineer32 FOREIGN KEY (ssn)
REFERENCES Employee (ssn) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE languages ADD CONSTRAINT FK_Languages33 FOREIGN KEY (ssn)
REFERENCES Secretary (ssn) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE Order ADD CONSTRAINT FK_Order41 FOREIGN KEY
(Department_name) REFERENCES Department (name) ON DELETE NO ACTION
ON UPDATE NO ACTION;

ALTER TABLE Order ADD CONSTRAINT FK_Order40 FOREIGN KEY (name)
REFERENCES Supplier (name) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE Employee ADD CONSTRAINT FK_Employee36 FOREIGN KEY
(Department_name) REFERENCES Department (name) ON DELETE NO ACTION
ON UPDATE NO ACTION;

ALTER TABLE Employee ADD CONSTRAINT FK_Employee26 FOREIGN KEY
(Employee_ssn) REFERENCES Employee (ssn) ON DELETE NO ACTION ON
UPDATE NO ACTION;

ALTER TABLE telephone_numbers ADD CONSTRAINT FK_telephone_numbers35
FOREIGN KEY (name) REFERENCES Department (name) ON DELETE NO ACTION
ON UPDATE NO ACTION;

ALTER TABLE Secretary ADD CONSTRAINT FK_Secretary34 FOREIGN KEY (ssn)
REFERENCES Employee (ssn) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Appendix D1: Steps involved in creating model and generating DDL-code in ArgoUML (and uml2sql)

This section describes the steps involved in creating the model and generating the DDL-script. The model is created in ArgoUML and has to follow a set of rules (explained below), while the DDL-code generation is accomplished through uml2sql, a small Java-based tool. It should also be noted that the steps described below is one of a number of ways to accomplish the task.

Creating the model

Step 0

Most computers use the java virtual machine (JVM), if not for standalone java-application, than for web-based programs (applets). Some installations of JVM might not include the jar tool (which is needed later in the study), so the version used here is Java 2 SDK version 1.4.1 (including the runtime environment). JVM is needed to execute ArgoUML and uml2sql.

Step 1

The first step is to create a class diagram (which is created automatically when opening an empty document). This is done through the “ClassDiagram” –option in the “Create diagram”-menu.

Step 2

A package is created through the package button in the toolbar. The package has to be stereotyped as “database” or “catalog” (see below for explanation), and the name of the package will become the name of the database in the DDL-code. A number of “tagged values” (name-value pairs that can be added for any type of structure in the model) has to be added to the package in order to make the generation work (these are explained below), this is done through the properties window for the package.

Step 3

In this step the tables (classes) are created. A number of tagged values are also added for keys (as explained below).

Generating the DDL-file (using uml2sql)

The model has to follow a set of rules to make the generation work, including certain tagged values. This is taken from the tool website¹⁰:

A class diagram is considered a database catalog if it is stereotyped <<database>> or <<catalog>>. Other models will not be processed. The class diagram can be stereotyped itself or an embedded package can be used in the class diagram (as is the case with this study). To specify the connection to the SQL database, the following tagged values has to be supplied:

¹⁰ <http://uml2sql.sourceforge.net/README.html>; accessed May 2003

Keyword	Explanation
Driver	Fully qualified class name of the JDBC driver (e.g. org.gjt.mm.mysql.Driver)
ConnectionString	The JDBC connection string to the corresponding catalog on the SQL database.

Table 11: Required tagged values for the database (package or model)

There are also a number of optional tagged values that are used for authentication:

Keyword	Explanation
User	The user name to log into the database. If not given, UML2SQL will prompt for it.
Password	The password to log into the database. If not given, UML2SQL will prompt for it.

Table 12: Optional tagged values for the database

Inside the stereotyped package classes, which represent tables, are drawn. A table is described by its name and its attributes. Currently there is no support for stored procedures so the operations don't do anything. Tables have the following optional tagged values:

Keyword	Explanation
PrimaryKey	The column list of the primary key.
Index	The column list of the index with or without index name.
Unique	The column list of the unique index (candidate) with or without index name.

Table 13: Tagged values used with tables

Steps involved in generating the DDL-code

Step 1

After saving the ArgoUML project file, it as to be unpacked (in order to extract the XMI-file). This is done with the following command (executed in a dos or linux prompt):

```
>jar -xvf filename.zargo
```

The XMI-file is usually named: filename_.xmi.

Step 2

This step includes configuration of uml2sql and installation of external programs. There exist a GUI (graphical user interface)-version of uml2sql, but it appears as if it is not working properly (it could not be executed); this is discussed in chapter 6. The text-based version is used instead.

D1 Steps involved in creating model and generating DDL-code in ArgoUML

The tool requires Xerces, the apache XML parser, version 1.1.0 or higher. Version 1.4.4 is used here. The `uml2sql.bat` (windows based systems) or `uml2sql` (Unix or Linux systems) needs to be edited, where the following variables are changed:

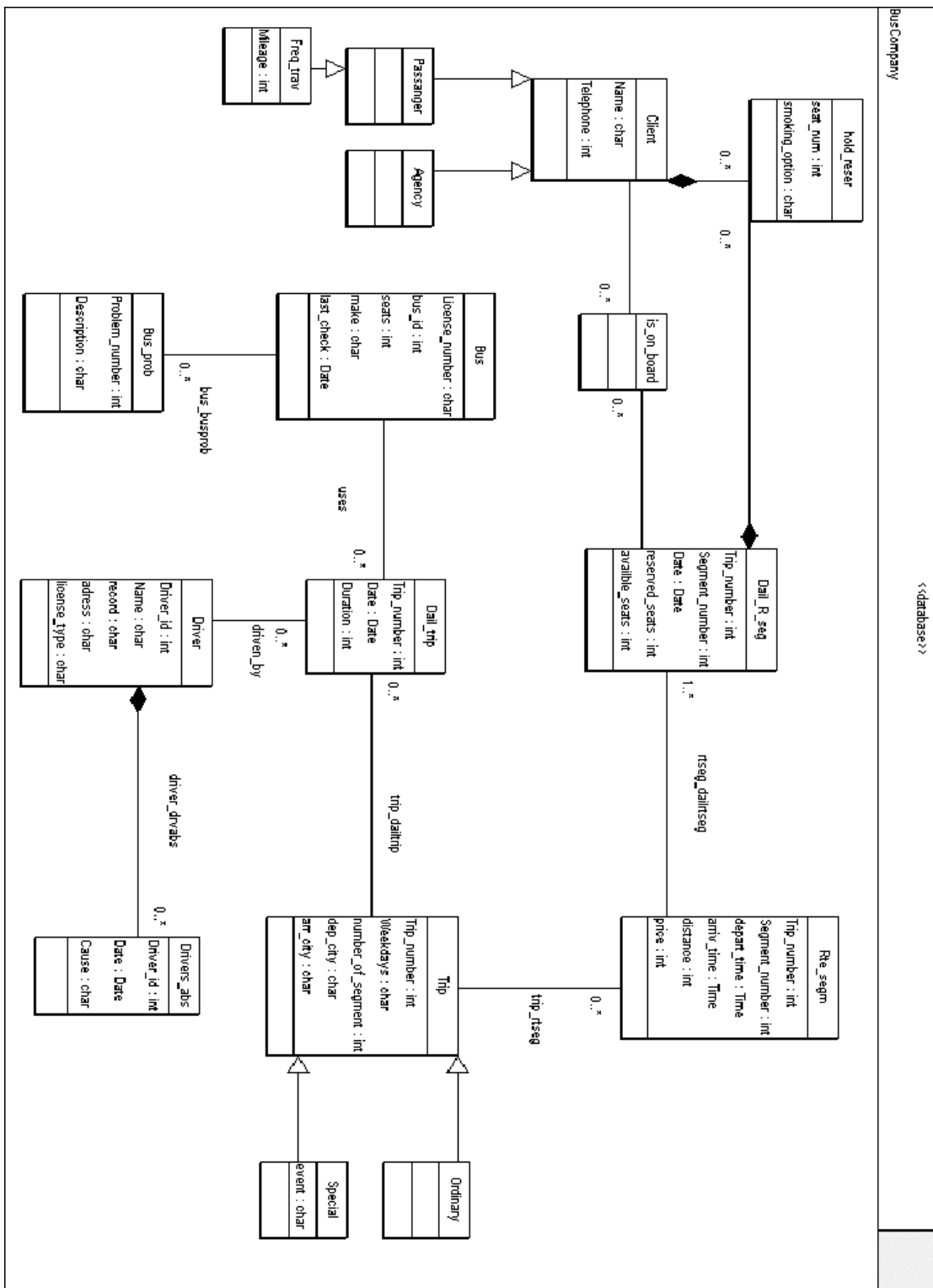
- `PATH_XERCES`: Full path to `xerces.jar` (the Xerces binary)
- `PATH_UML2SQL`: Full path to `uml2sql.jar` (the `uml2sql` binary). This variable does not need to be changed if `uml2sql` is run from the installation directory.

Step 3

The DDL-code is generated with this command (into dos or Linux prompt):

```
>uml2sql filename.xmi
```

Appendix D2: ArgoUML data model



Appendix E: Reverse engineering in Rose

This section describes the steps involved in reverse engineering (creation of model from DDL-code) with Rose. This procedure has been performed on both the Rose DDL-code and the PyDBDesigner DDL-code; the resulting models are also presented here.

Steps involved in performing reverse engineering in rose

This is a relatively simple procedure, where a “wizard” is used.

Step 1

The wizard is accessed through selecting the “Reverse engineer...”-option in “Tools” – “Data modeler”.

Step 2

After following the wizard, which includes selecting input DDL-file, a new schema is created automatically in the tree-view (usually placed to the left). To visualize the components of the model a new diagram has to be created, this is done through right clicking on the schema symbol in the tree-view and selecting “Data modeler” – “New” – “data model diagram”. After creating the diagram each component has to be dragged from the tree-view to the model canvas (relations are drawn automatically).

Model from PyDBDesigner DDL-code

Reverse engineering from this DDL-file was unsuccessful; the model created contained only two tables and errors were found in the DDL-code by Rose.

