A scalable repository based on a meta-modelling architecture

Master Thesis

By: Josbert Lonnee (s9801650)
Date: August 2008
Study: Master Computer Science
Track: Software Engineering

University of Twente
Drienerlolaan 5
Postbus 217
7500 AE Enschede
Netherlands

BiZZdesign B.V.
Colosseum 21
Postbus 321
7500 AH Enschede
Netherlands

BiZZdesign

Supervisors
Dr. Ivan Kurtev (primary)
Dr. Ir. Klaas van den Berg

Supervisor
Dr. Ir. Harm Bakker

Abstract

BiZZdesign is a company that develops tools for business processes and architecture modelling. With these tools customers handle models that conform to changeable meta-models. BiZZdesign currently supplies its customers with a repository that stores the models as atomic units. The meta-models can not be stored. Operational problems appear when models get too big. Moreover the repository can not interoperate with other tools. In this thesis we address the enhancement of this repository for:

1. Storing the *meta-models* to which the models conform;
2. Allow the connecting tools to be *scalable* with respect to the size of the models;
3. Allowing it to *interoperate* with other tools than that of BiZZdesign.

For the first and third point BiZZdesign could apply Model Driven Engineering (MDE) in order to obtain a meta-modelling architecture and communication standard. However existing approaches such as the Meta Object Facility (MOF) in the Model Driven Architecture (MDA) and the Eclipse Modelling Framework (EMF) do not supply a proper solution to all three above enhancements. Especially scalability is not addressed. On the other hand BiZZdesign's current technology especially does not address interoperability and how to store meta-models. In this thesis we evaluate all approaches and analyze their shortcomings. There is no standard that allows BiZZdesign to keep its particular object-oriented paradigm. Stop using this would be an unaffordable effort for BiZZdesign. Henceforth no standard or framework is used and we decided to build a repository from scratch.

In the context of MDE in general and BiZZdesign in particular we first provide a *novel system representation* for representing any model or meta-model and any of the relations between them. Beside meta-models must be represented and stored by the repository, we also investigate what we can store in and express with a meta-model. We describe how it can completely define an information system and thereby facilitate *pure* development based on MDE. We show how any information would fit in a meta-modelling architecture with three typing-layers and three meta-layers.

We achieve scalability by building a distributed system with client-server architecture. The repository contains all data and the tools that work with it have a limited view on it. A tool dynamically changes its view by loading and unloading clusters of data. When clusters are loaded, what they contain and what their uses are is specific for the tool's domain. Henceforth this is to be defined by the meta-model that is both the tool's definition and part of the repository's content. Our scalability solution is similar to that of the World Wide Web. While the data on the Internet does not fit on any single machine, a browser can still handle all reachable data by handling it on a page basis.

We incorporate our scalability solution in a design of BiZZdesign's repository and a prototype. For the first we in fact provide a design for a new product in the *product line* of BiZZdesign. It is based on a three-layer meta-modelling architecture that seamlessly joins BiZZdesign's specific OO paradigm. The prototype contains a repository and a tool and partially proofs our system representation's capability of representing all desired information and our scalability solution. Scalability is demonstrated by *benchmarking* an original tool of BiZZdesign and the tool of the prototype. Real proofs of our solutions are postponed until the actual development of BiZZdesign's repository that will occur after our work.

Preface

With this we describe our work between November 2007 and August 2008. Officially this work is internal for the University of Twente. Still it is executed on request of the BiZZdesign Company and is mostly executed there.

This thesis has a tree structure of depth three. The content is first divided in **chapters**, second each chapter is divided in a number of **section**s and third each section may contain a number of **subsection**s. The terms used here for the depths are used stringently throughout the entire thesis. All chapters but the first and the last start with an introducing section and end with a concluding section.

We would like to **acknowledge** Alfons Laarman for the term 'model space'. At explaining him what it was, he started to use this term from the moment it became clear to him. Thereby he implicitly indicated this term covers the idea behind it the best to the 'outside world'.

List of Abbreviations

Abbre-viation:	Full Description:
AO	Aspect-Oriented or Aspect-Orientation.
AOP	Aspect-Oriented Programming
API	Application Program Interface
CORBA	Common Object Request Broker Architecture. A standard of the OMG.
CVS	Concurrent Versions System
CWM	Common Warehouse Model. A standard of the OMG.
DLL	Dynamically Linked Library. Part of a program on the MS Windows platform.
DSL	Domain Specific Language
DTD	Document Type Descriptor. Document to which an XML file can conform.
EADF	Extendable Applications Definition Framework. Described in this thesis.
ECore	The meta-meta-model of the EMF.
EMF	Eclipse Modelling Framework [55]
GMF	Graphical Modelling Framework [54]. Also of Eclipse.
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IDL	Interface Description Language
JAR	Java Archive. A compressed file used in conjunction with the Java platform.
JRE	Java Runtime Environment
LDAP	Lightweight Directory Access Protocol. Is typically used to access a directory of users' information.
MDA	Model Driven Architecture™ [26] An approach to MDE proposed by the OMG.
MDD	Model-Driven Development. Typically used by Atkinson & Kühne [4] to indicate MDE. Synonym to MDE.
MDE	Model Driven Engineering
MDSD	Model Driven Software Development. Synonym to MDE.
MML	Meta Modelling Language. One of BiZZdesign's languages for expressing meta-models.
MOF	Meta Object Facility [35] A standard of the OMG.
MS	Model space.
MVC	The Model-View-Controller architectural pattern.
MySQL	A specific database platform [39].
O/R	Used when discussing an Object–Relational mapping or a tool therefore.
OCL	Object Constraint Language. A language for expressing constraints on objects.
OMG	Object Management Group. A consortium developing relevant standards.
OO	Object-Oriented
OS	Operating System
OSGi	Open Services Gateway initiative. Obsolete name for alliance creating open standards.
OWL	Web Ontology Language. The abbreviation is intentionally wrong.
PIM	Platform Independent Model. A type of model within the MDA approach [26].
PSM	Platform Specific Model. A type of model within the MDA approach [26].
RCP	Rich Client Platform [34]. Like the EMF and the GMF of Eclipse.
RTF	Revision Task Force. Involved in revising a standard.
SQL	Structured Query Language
SWT	Standard Widget Toolkit. An open source Java UI toolkit.
UI	User Interface
UML	Unified Modelling Language. Language for OO modelling of information systems.
URI	Uniform Resource Identifier
WWW	World Wide Web
XMI	XML Metadata Interchange. A standard of the OMG.
XML	Extensible Markup Language. Text based language for expressing general data.

Table of Contents

List of Tables

Table of Figures

1 Introduction

The context of our work is sketched by BiZZdesign that desires a model repository and ends up in at the MDE because of that. A first repository design is created, but some problems persist. A scalability problem appears to be the most prominent and pervasive. Our work continues on this 'runaway' design problem. We analyse a lot of MDE, which is currently in full development. We conclude with a repository design based on a proven scalability solution, but not based on any standard.

1.1 Context

There is a "recent trend" [27] to Model Driven Engineering (MDE). The MDE mostly concentrates on creating software applications by creating models. The concept of meta-model is central [29] in the MDE. Even standards like the Meta Object Facility (MOF) [35] are being developed for this area.

BiZZdesign Tools is a part of the BiZZdesign Company [8]. BiZZdesign Tools (from now on just called BiZZdesign) develops and maintains modelling tools. The current tooling is based on code originating from the Testbed Project [53]. The most important tools are:

- **BiZZdesigner:** Tool for modelling business processes. The models conform to a language called Amber [14];
- **BiZZdesign Architect:** Tool for modelling business architectures. The models conform to a language called ArchiMate [30].

Summarized these tools of BiZZdesign are typical modelling tools. Models are created, their information is graphically represented, information can get modified, etc. MDE was not yet a major development when BiZZdesign started. But currently the engineering at BiZZdesign has overlap with MDE. A key difference is that BiZZdesign's tools handle business models instead of software application or design models. On the other hand BiZZdesign does use meta-models like Amber [14] and ArchiMate [30]. As a result BiZZdesign followed this recent trend to MDE.

Next BiZZdesign also develops and maintains repository functionality. The current repository is conceptually a storage facility of only models as handled by the above tools. Physically it is a database. The customers of BiZZdesign can access a repository instance by using the tools. Next to storing models the repository functionality includes offering an overview of what is stored in the repository.

1.1.1 Previous Work

Before our work started BiZZdesign had formed new requirements for its repository functionality. Moreover efforts were made on designing a repository as a new product. This repository design has been the subject of our trainee project, which was executed at BiZZdesign and took three months.

As explained above BiZZdesign's current repository offers an overview of what is stored in the repository. Thereby BiZZdesign was in fact anticipating on offering a **workflow support** for the work on models by BiZZdesign's users. Part of the work of our trainee project concentrated on designing an improved workflow support for the repository product. In the rest of this thesis this is referred to as the **prior workflow support**. Another part of the work concentrated on creating a web based portal based on an adapted version of existing code. The first work on a high-level system design for the new repository product was described in [32].

BiZZdesign had put three more requirements for the new repository product:

1. The storage and communication of model information should conform to standards to facilitate **interoperability**;
2. Beside storing model information the repository should also store associated **meta-model** information;
3. The current tools of BiZZdesign are not scalable with respect to the size of models. The repository should solve this **scalability problem**.

During our trainee project there appeared not to be enough time to make the design meet these requirements. They appeared to be difficult and intertwined. Especially *the scalability problem appeared to be pervasive* and affect all aspects of the design and the implementation. They became the motivation for the work behind this thesis.

1.1.2 The Prior Repository Design

The previous subsection explains a basis for a design of a new repository has been created. Our work starts with this design and it is considered input. It is from now called the *prior design*. The main goal behind the prior repository design is to form a storage and registration of all artefacts of a customer of BiZZdesign. These artefacts include:

1. Business models;
2. Other artefacts related to business models;
3. The relations between all of business models and the other artefacts.

The design is different from the current repository as the latter only stores business models. The prior design covers:

- A repository as new product;
- How tools communicate with the repository;
- The tool's internal model data management.

According to the prior design the repository and tools conform to the client-server system architecture. An important aspect of the design is that it takes the efforts that BiZZdesign would need to create an implementation in the future into account. Design rationales are based on:

- Reuse of existing implementations;
- BiZZdesign's practical, internal experience;
- BiZZdesign's experience with its customers.

1.2 Problem Statement

Our work behind this thesis continues the design of the repository. We respect the prior design and its design rationales. We respect the insights previous work gained on the workflow support as well. The design must join the current tooling and functionality of BiZZdesign as most as possible and existing system parts and implementations must be reused whenever this is possible. Otherwise BiZZdesign's experiences are ignored. Moreover the future implementing of the repository can get too costly.

The previous section explains there are three remaining requirements the prior design does not anticipate on. These problems persist in both the current situation and this prior design. Hence these are the problems in stock for our work. We detail them in the next three subsections. In the remainder of this thesis these problems are respectively referred to as the **interoperability**, **meta-model** and **scalability** problems.

1.2.1 Interoperability

The current repository of BiZZdesign is completely customized to internal needs. A resulting interoperability problem is that only BiZZdesign's tool can interoperate with it. It is for example not possible to use it with MOF compliant tools of other tool vendors.

In the past BiZZdesign gained experiences with the software of Adaptive [1], which offer a repository solution that complies with the MOF standard [35]. In spite of the last it is yet impossible to let BiZZdesign's tools communicate with the repository of Adaptive as BiZZdesign's tools do not comply with the MOF standard.

Clearly interoperability should result from making the repository comply with a standard like that of the MOF [35]. The *interoperability problem* of the prior design is that it is unclear if and how the repository can comply with the MOF standard or another standard, while still meeting all other requirements.

1.2.2 Meta-Modelling Architecture

The current tools of BiZZdesign allow their users to change the contents of the meta-models to some extent. Hence the meta-models are members of the set of a customer's artefacts

that are related to that customer's business models. Thereby the repository must store the meta-models as can be concluded from subsection 1.1.2. Another reason for storing meta-models comes from anticipation on the interoperability problem. Hence there are two reasons for the repository to store meta-models.

By storing meta-models the new repository will be based on some meta-modelling architecture like we describe in subsection 2.11.7. The *meta-model problem* of the prior design is that it is unclear what meta-modelling architecture is able to shelter all models, meta-models and other artefacts the repository must store. Furthermore it is not clear if any standard, like for instance that of the MOF [35], defines an architecture that suits all needs.

1.2.3 Scalability

The current tools of BiZZdesign are not scalable with respect to the size of models. When a model's size gets above a certain system-dependant threshold the tools get unable of properly handling them. As we explain above a requirement for the repository is to solve this problem for the situation where a tool approaches a model via the repository. The *scalability problem* of the prior design is that there is no scalability solution.

1.3 Approach

The next subsections describe the steps of our approach to our problem. For each step we first describe the reason and then its contents.

1.3.1 Analyze the MDE

To solve the meta-model problem knowledge is necessary of modelling, meta-modelling and meta-modelling architectures. To solve the meta-model and interoperability problems knowledge is necessary about MDE related standards. Hence we study the concepts behind the MDE and related standards.

BiZZdesign's tools handle business models instead of software application models. Thereby we only need knowledge of certain aspects of the MDE. These aspects for example include:
- Meta-modelling;
- Model representation;
- Model communication.

Part of the literature we study is about how to approach the software application creation problem. An example is the Model Driven Architecture (MDA) approach as described by Kleppe et al [26]. Other literature is formed by standard specification like that of the Meta Object Facility (MOF) [35]. Finally we study much literature about modelling and meta-modelling in the light of the MDE.

1.3.2 Analyze BiZZdesign's Current State

As we mentioned in subsection 1.1.1 it is clear that our scalability problem affects all aspects of the design and the implementation. On the other hand part of our problem is that we must not divert too much from the prior design. This design also aims at reuse of existing implementations of BiZZdesign. This makes the current situation of BiZZdesign important for us. Hence the next step in our approach is to study the current situation around BiZZdesign's repository.

The current situation around the repository of course includes the storage of models. But other important subjects are:
- The way BiZZdesign currently expresses its meta-models;
- How the current tools internally represent models;
- What tools do with models and model data;
- The communication between the tools and the repository.

1.3.3 Design a System Representation

The current repository of BiZZdesign only stores business models. Yet the new repository must also store other artefacts including meta-models. Moreover also all relations between any couple of stored artefacts must be stored. We describe this in subsection 1.1.2. In this

step we first analyse what data must be stored. We also look at how this data interconnects by looking for example at the following kinds of relations:
- ❑ Between models as required for the workflow support;
- ❑ Between models as practiced by BiZZdesign's current tooling;
- ❑ Between models and meta-models;
- ❑ Within meta-modelling architectures.

In subsection 2.11.3 we will describe how a system benefits by a *system representation* that is as simple as possible and still capable of expressing all information that the system is about. As it is now clear *what* the repository must store we conclude this step by designing a completely new *system representation* for the information of the repository.

As antithesis we also anticipate on the fact that the storage is involved for the scalable model handling of the next step in our approach. Hence the next step in our approach has a *back-influence* on this step. This means the system representation must facilitate complete freedom of creating views on the stored content. We design a system representation that is simple and homogeneous, while it is able to represent any information the repository will need to store now and in the future. We refer to this system representation as the **model space**.

1.3.4 Design a Scalability Solution

The previous steps form all of the analysis and design for the interoperability and meta-model problems. The last problem that remains is that of the scalability. Hence the next step in our approach is to design a **scalability solution**.

First we analyze what scalability really means for modelling tools that work with the repository. In fact this analysis is done in parallel with the previous steps of our approach. We finally describe scalability of *interconnected data handling systems* in section 2.10.

We conclude the repository and the tools that work with the repository together in fact form a **distributed system**. According to our solution the tools do not load models that are isolated chunks in the repository. Instead the tools have a limited **view** on a single, 'huge', homogeneous storage. This view is dynamically changed by loading **clusters** from the repository and unloading. We describe a cluster is factually a subset of the repository's data formed and loaded for a specific use within the tool. We conclude that the freedom of forming clusters must not be impeded to get optimal scalability. We next conclude an (internal) system representation can impede this. This way this step of our problem approach has a back-influence on the previous step in our approach. We describe this previous step and mention this back-influence in subsection 1.3.3.

1.3.5 Repository Design

Now all problems with the prior design are analyzed and we have enough knowledge of BiZZdesign's current situation we carry our knowledge through in the repository design. We create a repository design that is subsequent to the prior design, but without its problems. The design encapsulates the scalability solution of the previous step in our approach. The system architecture is the client-server architecture. The server conceptually stores the model space. This model space contains a meta-modelling architecture that is custom to BiZZdesign.

A major issue for this step of our approach is to let the design join the current tooling and functionality of BiZZdesign as described in section 1.2. Hence we respect the prior design and insights on workflow support. This includes the respecting of the choice for the client-server architecture.

As BiZZdesign's current tools for instance share code and components they in fact form a product line. The repository will be a next product in this line. Hence the design process in fact borrows down to "software **product line engineering**" [42]. The actual development of the repository will be done after our work behind this thesis and in lock-step with the current development steps.

1.3.6 Building a Prototype

At the previous step of our approach the scalability solution is based on clearly defined concepts. The fact that tools that work with the repository have a limited view leaves the following questions open:

- Can they work with their view?
- Can they update data while keeping the repository's data consistent?
- Tools can dynamically expand their view. Will this view not get too big?

We think these questions can only really be answered by a real implementation of the solution. It is foreseen that the development of the repository according to the refined design will take BiZZdesign a couple of years. Hence this will not happen as part of our work. Still we want the above questions answered before this happens. Therefore we build a **prototype**.

This prototype will be used for the comparison that is the next step of our approach. Therefore the prototype:

- Offers functionality that is representative for a part of the functionality BiZZdesign currently offers to its customers;
- Implements the scalability solution;
- Has a system and meta-modelling architecture that is similar to the refined repository design.

The last two months of work were spent on the creation of this prototype.

1.3.7 Compare Using a Benchmark

By only building the prototype the scalability solution is not evaluated. Hence we use a **benchmark** to compare the prototype with current technology of BiZZdesign. Both are tested with 'huge' amounts of data. Their behaviours are monitored and observations form the results.

1.4 Contributions

Our work contributes to both BiZZdesign and to the development of MDE supporting tools in general.

1.4.1 Investigation of Eclipse's Frameworks

Eclipse offers much software and many frameworks and standards. Eclipse aims at supporting software development. It is clear the aimed development has much overlap with the development as BiZZdesign. This overlap includes for example:

- **Tool Building:** BiZZdesign develops tools that are based on a rich UI. Eclipse offers the Rich Client Platform (RCP) [34] to support this development;
- **Meta-Modelling:** BiZZdesign develops meta-models and tools that handle models conforming to these meta-models. Eclipse offers the Eclipse Modelling Framework (EMF) [55] to support this development;
- **Graphical Model Representation:** BiZZdesign develops tool that graphically represent parts / aspects of models. Eclipse offers the Graphical Modelling Framework (GMF) [54] to support this development.

The mentioned support of Eclipse requires development in the Java language. For BiZZdesign it would be a tremendous effort to port all their code to Java. Therefore we contribute to BiZZdesign by investigating how useful Eclipse's contributions are to BiZZdesign.

We do not only look at BiZZdesign's current state. We also investigate how useful Eclipse's offer would be for the future repository. We focus on the EMF's capabilities of expressing meta-model architectures and solutions for scalability.

We conclude Eclipse's offers should not (yet) be used by the following:

- The EMF contains functionality for generating code on the basis of a meta-model. Yet this currently still is quite limited. Still many parts of the generated application must be coded manually;
- The EMF itself offers no scalability solution. It only allows an application that uses it to be set up scalable, but only in some specific way;

- The GMF is yet not mature enough to be used in a production environment as that of BiZZdesign;
- The meta-model a model conforms to and graphical representations of that model should be separated. Still the GMF generated graphic representation code based on assumptions about the meta-models.

1.4.2 A View on MDE

As the introduction describes there is a "recent trend" [27] to MDE. The fact that it is a trend means there currently is a lot of research to it. The fact that it is recent means there currently is not much consensus on shared key concepts. For instance there still is much debate around the central concept of the meta-model. Gogolla et al [20] also mention that "notions within the metamodeling area are often loose due to a lack of formalization. This has been recently referred to as the meta-muddle."

We contribute by giving a view on the concepts of MDE. The *meta-modelling* concept is central. We focus on how we think meta-models define information systems and how meta-modelling architectures are formed. Moreover we gain more insights on the MDE in general and the data aspect of the models and meta-models in particular by designing our scalability solution, BiZZdesign's repository and our prototype. In chapter 11 we describe what we learn.

1.4.3 Repository Design

We contribute to BiZZdesign by creating a full-scale design for a future repository. This design is based on the prior design, joins the current tooling and functionality of BiZZdesign and incorporates the new scalability solution. This solution is based on clearly defined formalisms. The creation of the design aims at reusing as much as possible of BiZZdesign's current technology. Except for creating a repository design our work behind this thesis did not contribute to further phases of the development. For instance we do not create and test an implementation of (a part of) the final repository.

1.4.4 A Proven Scalability Solution

We contribute to the development of MDE supporting tools with a scalability solution. We base our solution on our observations on how applications handle data. For our solution we consider the repository and the tools that use it as a typical distributed system. We have built a prototype as part of our problem approach (see subsection 1.3.6). This distributed prototype system proves the solution really allows a model data handling tool to work with a 'huge' amount of model data. Still this amount is allowed to contain any desired amount and kinds of interconnections.

1.5 Outline of the Thesis

The background of our work is given shape by the prior design, our investigation of eclipse's frameworks, the research we did with the first two steps of our problem approach and our general research to data handling. In chapter 2 we describe the basics of this background. Chapter 3 gives more details of our investigation of Eclipse's frameworks. We postpone describing our relevant findings about how systems handle data in general and distributed systems in particular till chapter 7.

These further details are based on experience gained by experimenting with and using the frameworks. This experience is one of our contributions and is therefore not described as part of our background in chapter 2.

Next follows the result of our research to MDE in general. Chapter 4 describes the first step of our problem approach. It describes the system representation we design for the repository. The latter can then represent any model information. Chapter 5 contains our contribution of our view by focussing on meta-models and how they model an application. In chapter 6 we extend this view as we focus on meta-modelling architectures.

Chapter 8 summarizes our scalability solution and gives the underlying definitions. It is mainly based on our vision on data handling we give in chapter 7. Next chapter 9 contains our repository design that incorporates that solution. The architecture of the system and the meta-

modelling architecture are both described together with their integration. The physical storage gets highlighted as well.

Chapter 10 shortly discusses the design and implementation of the prototype we created at the one-but-last step in our problem approach. The last step in our approach is benchmarking our scalability solution. The description of this benchmark and the results form the last part of chapter 10.

Figure 1 depicts the chapters of our thesis and their dependencies. For the chapters 2 to 10 transitivity mostly applies. The back-influence from our scalability solution on our system representation we mention in subsections 1.3.3 and 1.3.4 is also shown.

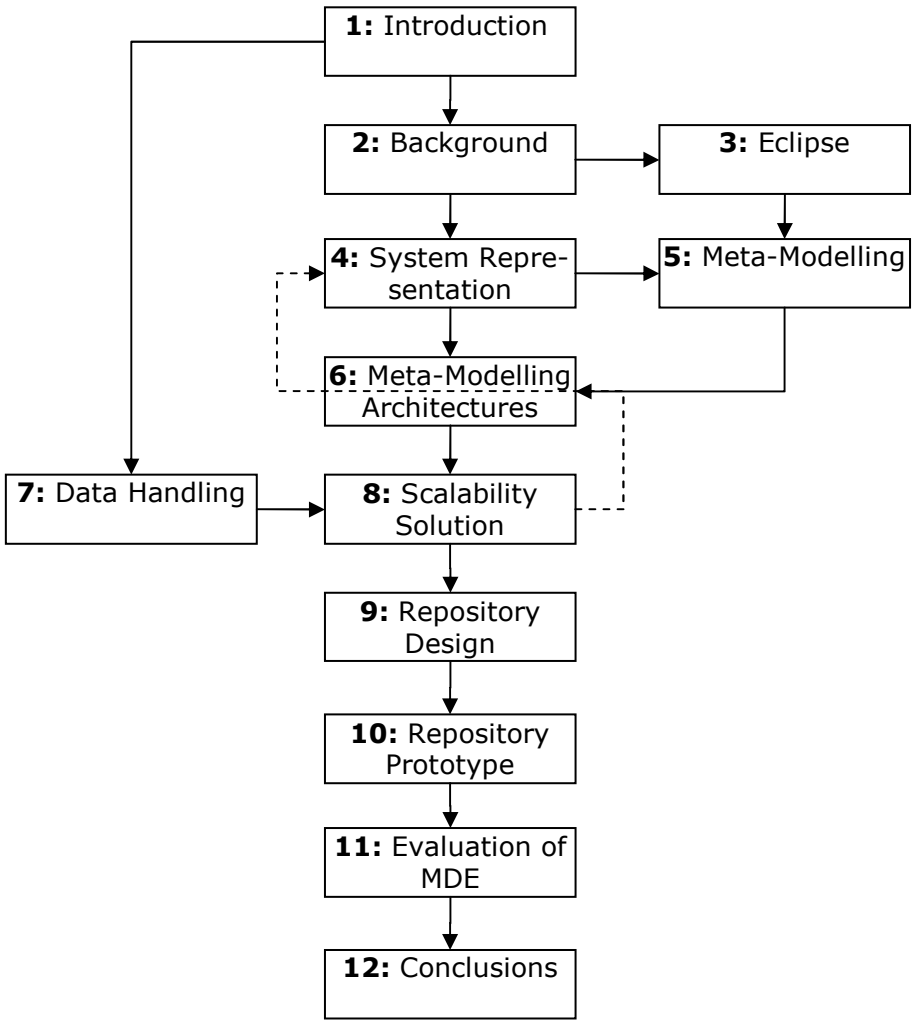

Figure 1: Dependencies between the chapters.

The chapters 3 to 6 together form a first part about the MDE in general. The chapters 7 to 10 form a second part that focuses on building a scalable repository. The second part is based on the first part. In chapter 11 we evaluate MDE in general and describe what we learn about it in the second part. Finally chapter 12 is our conclusions chapter. There we evaluate to what extend we solve our problems. Furthermore we give a summary of our work, give recommendations and describe outlets for future work.

The chapters 2 to 10 are intentionally removed. They can be supplied and may be distributed, sold and/or copied only with permission of BiZZdesign B.V. in the Netherlands,

www.bizzdesign.nl

11 Evaluation of MDE

In the chapters 3 to 6 we create a custom view on the MDE in general, where we use a top-down approach. In particular we concentrate on the data aspect of models and meta-models. In the chapters 7 to 10 we design a practical, scalable repository solution, where we use a bottom-up approach. In this chapter we evaluate the MDE after what we learned from the second approach.

11.1 Introduction

As the new repository of BiZZdesign will be about models, meta-models and their relations we got to the subject of MDE. This way the research area of the MDE became a major part of ours. The MDE aims at development of information systems by creating models. On the other hand the repository is mostly about storage of data. Hence we mostly concentrated on the data aspects of the involved subjects. We used a top-down approach and found there is much unclarity at this aspect of the MDE.

The MDE finally gave us a good impression of *what* data BiZZdesign's repository will be about. On the other hand we had to find out *how* this data should be handled to get a scalable solution. Here we used a bottom-up approach. We found much of the unclarity can vanish by first giving good, simple and strict answers to basic questions like:

- What is in a model? How do we basically represent a model?
- What is in a meta-model?
- How does a meta-model relate to software?
- What is instantiation?

In section 11.2 we give general conclusions on and recommendations for some basic MDE concepts from our background. We observed that, at developing information systems, scalability issues typically have a back-influence on the design. With real MDE an information system is completely defined by a meta-model. Hence designing a system is creating a meta-model and scalability issues must be fed back into the meta-model during development. We describe this in section 11.3.

11.2 Basic Concepts

We conclude on how we see a model in the light of MDE and how we can express instantiation between models and the meta-models they conform to.

11.2.1 Modelling

The MDE aims at creating information systems by creating models instead of traditional code. We conclude that, in the light of MDE, *a model is just an information carrier*. It typically consists of the data of a system or information about an aspect or part of the design of a system. A model is typically not any more considered an abstraction of something of the real world as with the classical conception of a model. Still traditional implementation artefacts like code in particular are also information carriers. What are the benefits from creating models over creating code?

With MDE the models are explained to be typically abstract. Productivity benefits should come from the fact there is less information in the models and thus less need to be defined at developing an information system. On the other hand also more abstract specialized languages can be created that bring the same benefits with traditional coding. These benefits are for example brought by languages like the PHP for developing web services and the SQL for describing database communication;

An added value of a model over code we see is that it is not necessarily text. This *can* bring the advantage of not any more suffering from the "tyranny of the dominant decomposition" [52] that a grammar brings. Next people have proven to be more efficient at working graphically: *A picture is worth a thousand words*.

Kleppe et al [26] describe the Model Driven Architecture™ approach for MDE of the OMG. They even explain code as a model. On the other hand the Java development language is

known for its platform independence. Combining these two facts Java code describing some information system is a Platform Independent Model (PIM) of that system. Java has already proven itself at the development of a wide range of information systems. In our opinion the benefits of the MDA have this way faded.

While we observe that the term 'model' is, in the light of MDE, given a different meaning than the meaning from before the research around the MDE, we propose no alternative for this term either.

11.2.2 Matters of Meta-Modelling

Kühne [27] distinguishes two "Flavors of model and element instantiation": ontological and linguistic. "Ontological instantiation between two elements or models is […] based on the relationship between them in terms of their meaning." [27] "Linguistic instantiation between an element and a linguistic type is based on the assumption that the type represents a (fragment of a) language defining which expressions are valid sentences of it." [27] We observe from his work how linguistic instantiation is used to allow systems to approach information that is formed by instances. Next the types involved in this instantiation can be used to express an 'orthogonal' instantiation that can have any meaning for the system. Two linguistic type elements can for example be a 'class' and an 'instance'. The relation between an instance of 'class' and an instance of 'instance' then is what Kühne [27] explains as ontological instantiation.

But we distinguish three flavours of instantiation:
1. Each element of each model at any meta-layer or typing-layer is an element instance of the model space;
2. Each model space element has another model space element as type. This is indicated by the typing relation of the model space;
3. A meta-model can for example define a 'class' type and a related 'object' type. That makes it possible to express in conforming models how instances of 'object' are instances of instances of 'class'.

The way Kühne [27] describes his two flavours of instantiation gives the impression any instantiation is always one of the two. We conclude *the two flavours are roles instantiations have in relation to each other*. With the above instantiations the first is linguistic with respect to the second and the second is linguistic with respect to the third. Vice versa the third instantiation is ontological with respect to the second and the second is ontological with respect to the first.

Above we explain linguistic instantiation is used to allow systems to approach information that is formed by instances. This still holds. On basis of the first of the above instantiations the repository system approaches the model space. A system that corresponds to a meta-model in the repository approaches conforming models that are also in the repository on basis of the second instantiation. When the system that corresponds to that meta-model is a tool for creating UML models the system under development that is modelled with that tool will approach its data on basis of the third instantiation.

11.3 Back-Influence by Scalability on Design

We conclude on the practice of creating information systems by creating models. In our opinion an information system should, for complete MDE, be completely defined by the models, not only modelled. In this section we describe the back-influence of typical scalability problems in the system development process in terms of:
- Designing systems by creating a meta-model like we describe in chapter 5;
- How systems handle data like we describe in chapter 7;
- Our scalability solution we describe in chapter 8.

This section contains a reasoning of the correlation between the designing of an application's:
1. **Conceptual** storage of data;
2. **Practice** of data handling;
3. **Physical** storage of data.

In the remainder of this section we refer to these with (1), (2) and (3) respectively. The data concerns that of any application in general, but also the model and meta-model data of the repository and tools in particular.

11.3.1 Desired Development Practice

When designing an application the formulation of its (1) should be regardless of its (2) and (3). As its (1) corresponds to the domain the application is created for, the disregarding assures the freedom of creating applications for any desirable domain. Next at developing an application its (2) will be based on its (1) and thereby depend on this. We also describe this in chapter 7. An application's (2) can not always (completely) be derived from its (1). With our scalability solution the definitions of clusters' contents (defining the (2)) for instance form additional information of the meta-model (defining the (1)). Finally at the implementing of the application, its (3) should be derived from or based on its (2).

At developing the repository no assumptions about its (1) or (2) are on stock. The only thing known by the design is that it will have the form of cluster communication. As a consequence the repository's (3) must be able to allow any (2). The best information about the (2) is retrieved by looking at how tools approach the repository's data in general. This should not be evaluated once, but once in a while during the time the repository is in use. An example evaluation was in fact described in A.3.2, where indexes were added on tables regarding their (indirect) use by the prototype tool. Hence the (2) of the prototype emerged not before a representative version of a tool (i.e. the prototype) ran for the first time.

The above reasoning implies an ideal order for application development. Ideally an application's (1), (2) and (3) are developed in this order. When the application is a tool that works with the repository, it is not necessary to derive its (3) from its (2) as this is fixed by the repository's system architecture. Still the above ideal ordering is recognizable at the development of a tool that uses the repository. A tool's (2) still needs to be based on its (1).

11.3.2 Real Development Practice

As opposed to the previous subsection that is about the *desired* practice, we now describe the *real* practice of developing applications in general and tools that use the repository in particular.

Above we explain formulating an application's (1) have complete freedom, but is that attainable in reality? For developing an application's (2) it is clear there are *practical limits*. The handling of data by processes, algorithms, etcetera is typically considered unpractical when it just takes the application too long. There are also clearly *physical limits* for an application's (3). Typical examples are the limits on the amount of system memory, disk storage, network capacity, etc.

With the desired application development first its (1) is designed, then its (2) is developed and then its (3) is implemented in a single run. But with real development this run becomes a zigzag. When basing an application's (2) on its (1) or its (3) on its (2) results in a (2) or (3) that is feasible and satisfies all requirements, then the development is settled. But that result is not guaranteed. **When any resulting (2) or (3) is infeasible, this must out of necessity be fed back into the (1) or (2) it is based on respectively.** Without such back-influence all systems would probably have been designed for a 24/7 running main-frame with unlimited, random-access system-memory and unlimited communication without delays. Many examples can be given of the above described back-influence.

We can now illustrate this back-influence during application development by using our scalability solution. Now a tool that uses the repository for its (3) and the cluster communication for its (2) is the application. The repository's (3) can have a back-influence on a design of tool's (2). It is not expected that the amount of meta-model data in the repository will impose any (future) scalability problems. Still the amount of model data can. *Time will tell whether the repository's (3) and framework for (2) will suffice for all tools in the future*. Next, when a tool's clusters' definitions (i.e. its (2)) are designed this can back-influence the tool's (1): its domain concepts!

In section 8.5 we discuss what modification of model data means. Typically multiplicities that are practiced for extends of relation types have consequences for modifications. At creating a

meta-model defining a tool we desire the complete freedom of assigning each relation type any practiced multiplicity. But in that section we conclude these multiplicities have consequences for the an tool's (2) when they are any other than `0..* – 0..*`. (See subsection 8.5.3 for the notation of multiplicities.) When a tool handles model data that is structured according to meta-model data that includes such relation types, it is forced to *mind* the multiplicities at modifying that data. Moreover it can be forced to mind multiple relation types' multiplicities when these relation types are interconnected. The latter can force a tool's (2) to place multiple modifications in a transaction.

When a tool needs to edit a model element that has a relation with a type that has a multiplicity that has a left and right upper bound of at most one: `0..1 – 0..1`, `0..1 – 1..1` or `1..1 – 1..1`, then it must, for that relation, mind at least one other element. This can lead to the need for transactions to update the element. Still this does not necessarily lead to scalability problems. When the tool needs to edit a model element that has a relation with a type that has a multiplicity that has an unlimited upper bound: `0..1 – 0..*`, `0..1 – 1..*`, `1..1 – 0..*`, `1..1 – 1..*`, `0..* – 1..*` or `1..* – 1..*`, then it possibly needs to mind an arbitrary number of other elements. This can mean a tool is forced to mind more elements then practical limits allow. This possible scalability problem can get even worse when the meta-model specifies multiple interconnected relation types with such multiplicities. What for example if type A is related to B where the relation has a type with multiplicity `1..1 – 0..*` and B is related to C with a `1..1 – 0..*` relation type? If a tool needs to delete an element instance of A, then it must do this in a transaction that minds all instances of B and all element instances of C connected to each of these instances.

11.3.3 Acceding Back-Influences

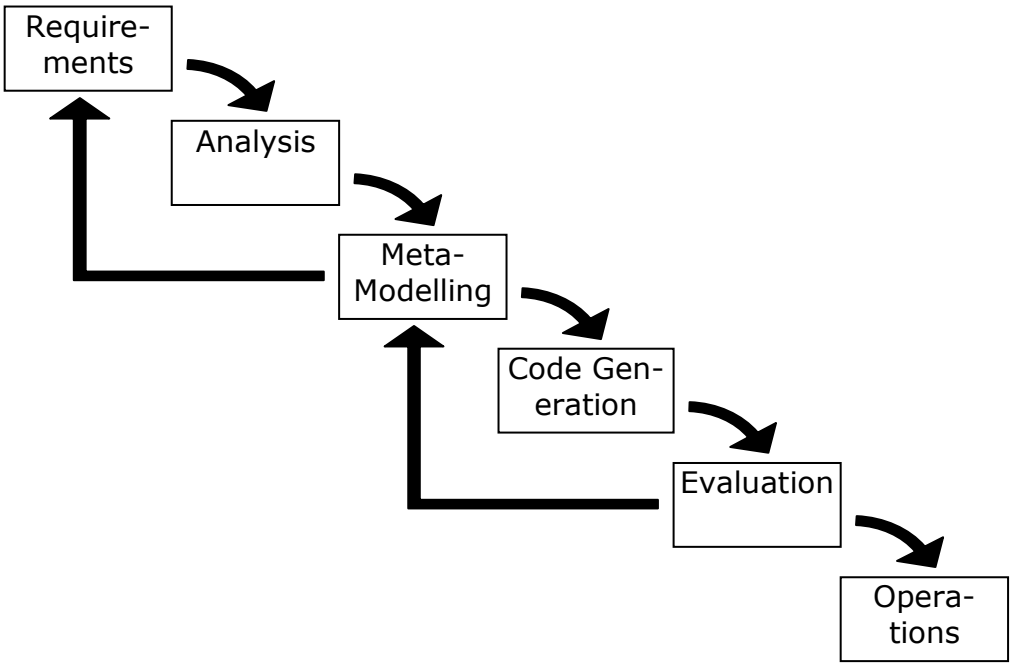

Figure 2: The waterfall model of MDE.

Above we describe where potential scalability problems arise from. Acceding to a typical back-influence on a meta-model is to change a relation type's multiplicities from `1..1 – 0..*` to `0..1 – 0..*`. A tool that is associated with the changed meta-model will relate all 'right' elements to exactly one 'left' element. Yet the change in the meta-model allows the tool to delete a 'left' element and then postpone the handling of the previously related 'right' elements. New requirements can be formed for the tool about what to do with unrelated 'right' elements. In summary after evaluating a generated system a change in the meta-model may be required. Next these changes may result in changed requirements.

Royce [46] gives a sequential software development model and explains it is an example of a flawed model. Others mostly refer to this model as the *waterfall model*. In Figure 2 we give a waterfall model that models the development process when based on MDE.

In our waterfall model in Figure 2 the design is replaced with meta-modelling. The implementation is replaced by code generation. As the code is generated is should not need any testing. Still this phase can not be removed and we call it the evaluation phase instead. "The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed." [46] Also scalability issues can not be experienced before this phase. At the evaluation phase problems can be experienced that one can not overcome by changing the implementation only: "Either the requirements must be modified, or a substantial change in the design is required." [46]. According to Royce [46] this results in back-influences similar as we depict in Figure 2. These are the same back-influences as we describe at the start of this subsection.

11.3.4 The MDA Promise

In section 2.5 we describe the Model Driven Engineering™ approach of the OMG. Kleppe et al [26] describe this approach as where first one Platform Independent Model (PIM) is created for an information system. Next this PIM is transformed into multiple Platform Independent Models (PSMs) in an automated way.

Above we describe how scalability problems can have a back-influence on the system model creation. Clearly these problems are platform specific. Hence acceding to such a back-influence means platform specific details need to be added or modified to the model. With the MDA this is not possible as a system designer can only make design decisions that are platform independent as he or she only creates a PIM. In other words we believe this approach is not suited for "development of large software systems" [46] as acceding to back-influences like described by Royce [46] is not possible.

In chapter 5 we describe it must be possible to model on the basis of platform independent design decisions as well as typical platform specific design decisions. Moreover we conclude platform independence is a relative concept. A central `Domain Concepts` part in a system's model is the most independent of any platform. Still even this part is not completely as it only applies to the platform of interactive systems. The more indirections via which another, supplementing part is connected to this part, the more platform specific it is. In terms of our view on creating system models and our scalability solution, acceding to back-influences by scalability problems means some `Cluster Definitions` or the central `Domain Concepts` model part is modified.

11.3.5 Workflow Support by Back-Influence

We believe the workflow support the repository is to offer in fact results from back influences as we describe above on requirements. We look at the practice of people working on repository content. There are situations where this practice shows limitations. Not at any time of this practice the content will be in a perfect, final state. This is especially true when the content is large and it is always approach on partial bases for scalability reasons. There will just be moments on which the content is in some intermediate, half-finished state. To get a practical system this necessity results in requirements that allow the content to be in such a half-finished state. Hence these requirements make the content subject of the entire workflow instead of end result of this flow only.

11.4 Conclusions

In the previous section we describe why the MDA can not realize its promise. Still we see benefits of MDE in general. When the practice of developing computer systems is changed from traditional development to MDE according to our view we give in chapter 5, this practice will significantly change. Traditional development is based on object-oriented (OO) or aspect-oriented (AO) programming and we can describe this traditional practice of developing applications roughly as:

1. Creating text-based code artefacts that conform to a programming language;
2. The code is partitioned according to one 'dominant decomposition' [52] in its textual representation, implied by the grammar of the programming languages;
3. As far as elements in the code do not interconnect according to the dominant decomposition, it interconnects by identifiers that are names and references to them;

4. Typically one language and paradigm is used for multiple parts or aspects of the developed system. Only the choice for identifiers and code comments indicate what artefact belongs to what part or aspect.

We can also describe the corresponding development practice according to our view on MDE:
1. Creating models that conform to a meta-model;
2. Assuming models are represented as a graphs there is not necessarily a 'dominant decomposition' [52] in this representation;
3. Assuming all model data is in one model space, then the models and all elements in the models uniformly interconnect by a single kind of relation;
4. Each model has a meta-model that is specialized to the purpose of the model. By this specialization the model can be as abstract and concise as possible; below we describe why. Names and identifiers should only originate from the domain of the developed system. The models should be clear enough to eliminate the need for (model) comments and design documentation.

According to our view on MDE we give in chapter 5 a meta-model fully defines an information system. Next a meta-model consists of multiple parts. Each part typically describes a part or an aspect of the system.

Describing the above otherwise each part of a meta-model is in fact associated with a concern of the system and as the parts are separated we have a "separation of concerns" [52]. This way we get to the world around Aspect-Orientation (AO). We conclude that the research behind *MDE uses a top-down approach* for separating concerns. The development behind MDE allows more and more concerns of an information system to be defined in models, which are next combined into one system where they are intertwined. The research behind *AO uses the corresponding bottom-up approach*! The development behind AO allows more and more concerns of an information system, that are intertwined at the system level and in using conventional programming languages, to be expressed separately.

While we observe the research behind both MDE and AO aim at a separation of concerns, there is also a difference. With aspect-oriented programming (AOP) a system can be developed by defining *any desired number of aspects*. All aspects are expressed using a single language like AspectJ. With MDE a system can be developed by defining *a fixed set of aspects with predefined roles and interconnections*. In chapter 5 we mentioned roles like `Domain Concepts`, `Constraints`, `Edit Operations`, etc…

With conventional programming and AOP the abstraction level can also be raised by developing abstracter or more high-level languages. Above we describe that with MDE we express each aspect of a system with a specialized model. This is the *one and only* reason why the abstraction level can be raised by the MDE with respect to conventional programming! When a meta-model part has for instance the role of constraints definition within a meta-model, then typically a language like the Objects Constraint Language (OCL) can be used when using MDE; when AOP is used these constraints must still for example be express in AspectJ, which is not as specialized to expressing constraints as the OCL is. Finally the benefits from raising the abstraction level are obvious. Also Atkinson and Kühne [4] concluded this: "By raising abstraction levels still further, MDD aims to automate many of the complex (but routine) programming tasks". [4]

The MDE subject is very broad and we describe above we think it is a good development. There are currently multiple views on and intentions with it by the width of this view. This is not necessarily a problem; all ideas and views can be united when they are based on defined, fixed and basic ideas and concepts. The different views on MDE can coexist by using different compositions of these basics. Of course this starts with a general agreement on these basics. Yet we have not found such an agreement. Even what models are, how they should be represented and what kinds of information they can contain is currently different in all views. Until this kinds of basics are not established the evolution of MDE is adrift. Gogolla et al [20] also discuss "notions within the metamodeling area are often loose due to a lack of formalization. This has been recently referred to as the meta-muddle." [20] Our formalization of the model space in chapter 4 might be a start of defining basics for the MDE.

12 Conclusions

BiZZdesign had interoperability and scalability problems with its model repository. Moreover it did not store any meta-model information. We did research to MDE and scalability for solutions. We created a new design and finally a prototype that partially proves out scalability solution. We gained many insights in data handling and MDE.

12.1 Current Repository

Currently BiZZdesign supplies its customers with repository functionality. This functionality is an intrinsic part of the tools BiZZdesign continuously develops. When users of these tools use the repository functionality this means that models that are handled by the tools are stored in a database. The tools directly connect to this database. The current situation has three problems we later refer to as the meta-model, interoperability and scalability problems. BiZZdesign previously formulated the goal to develop a new repository product that should solve these problems.

12.1.1 Problems

The models of BiZZdesign's tools have clearly defined meta-models. These meta-models are defined using a method that is particular for BiZZdesign. A core part of a meta-model is fixed. A part of the code of the corresponding tool is generated from this core part. Parts of the meta-models that extend this core part can be changed without changing the tools. These parts are loaded dynamically when the tool starts. With the current repository functionality is that these variable **meta-model** parts can not be stored. This can lead to problems when a BiZZdesign tool user loads a model from the repository that conforms to a meta-model that the user has no knowledge about or has not available.

With the current repository functionality the tools communicate with and store models in the database using a method that is custom to BiZZdesign. The only fully developed way of using and communicating with this database is by reusing the database scheme and code that is currently part of BiZZdesign's tools. The current repository does not comply with any standard for its external communication. In other words the current repository functionality lacks **interoperability**. To enable other tools that that of BiZZdesign to communicate with the repository database will lead to development problems as this is not trivial. An option is to reuse the part of the code of the repository from BiZZdesign's tools. But this would bind the development of the other tools to the same development platform as that of BiZZdesign's tools. Another option is to rebuild this communication, but this needs a lot of development for each individual tool.

With the current repository the models are stored as atomic units. When a tool loads a model from it, this model must be loaded entirely. Operational problems appear when models get too big. In other words the current repository functionality has a **scalability** problem with respect to the size of models.

12.1.2 Goals

Before our work started BiZZdesign has formulated the goal to develop a new repository. This new repository is not to form a part of the functionality of the tools, but a new product of BiZZdesign. Once it is a new product it must also store not only models like with the current functionality, but *any* information of customers of BiZZdesign that is relevant for describing and modelling their business. Next to forming a general storage, with the new repository product must also solve the problems we describe above.

Part of the work from before our work on this thesis resulted in a first design for the new repository product. We refer to this design as the *prior repository design*. This design follows client-server architecture, whereby the tools form the clients and a new repository application forms the server. This server exclusively handles the database communication. The communication between the tools and the repository application would be network based.

The new repository should solve the above problems and it must be possible to let other, new tools interoperate with it. At least a web portal is foreseen as one of these new tools. Scalabil-

ity should be a result of how the tools, including BiZZdesign's current tools, connect with it. Obviously also the meta-models should be stored as these are parts of BiZZdesign's customers' information describing their businesses. As BiZZdesign wants the new repository to store as much as possible of this kind of information, a question became: What can be stored in and expressed by a meta-model?

12.2 Research

In this thesis we tried to let BiZZdesign meet the goals of the previous section. Hence we addressed the enhancement of the prior repository design to make it solve of the above three problems. We also tried to give an answer to BiZZdesign's question of what can be expressed by a meta-model beside what BiZZdesign's meta-models currently do. Altogether this led us to do an extensive research.

BiZZdesign's current technology particularly has no solutions to the meta-model and interoperability problems. On the other hand no standard or framework is on stock that supplies us with a complete scalability solution. Support for creating a custom scalability solution is however sometimes included by frameworks, but these frameworks are too strict with respect to BiZZdesign. They would not allow BiZZdesign to keep using its particular object-oriented paradigm. In summary nothing supplies us with a solution to all three problems. We decided to design solutions from scratch. The communication with the repository complies with not any standard and no technology or framework is used according to our new design of the repository. We developed a system representation, a meta-modelling architecture, repository communication and a scalability solution based on basic principles.

12.2.1 Meta-Modelling Architecture

In chapter 4 we designed a novel system representation for the repository. In subsection 2.11.4 we explain what a system representation means in our opinion. It should be capable of representing any of the system's information. We finally developed the **model space** system representation in chapter 4. Within a model space all information is represented with a set of elements, a single binary relation and a single typing relation. The resulting, special *graph* is capable of representing any model, meta-model and any relations between them. Why it is capable we grounded philosophically in chapter 4. Indirections can be used to represent complex information with this simple representation.

To answer the question of what can be stored in and expressed by a meta-model we described how meta-models define an information system in chapter 5. Full MDE would be attained when a meta-model forms a complete definition of such system. We observe a meta-model consists of multiple models. A central domain concepts model indicates what information the system is about. This model is supplemented with models that define an aspect or part of the system. A model can also be a specialization of another and contain more detailed definitions that are typically specific for certain functionality or platform. We observe that platform independence is a relative concept. A domain concepts model is the most platform-independent; still it is not completely as it for instance only applies to the platform of interactive systems. Together with all supplementing models it forms one interconnected unity. The more steps a model is away within this unity, the less platform-independent it is.

In subsection 2.11.4 we also explained that a system representation should be simple to reduce the complexity of that system. Next the simplicity of the model space made our problem of creating an ideal meta-modelling architecture simple too. In chapter 6 we applied the view of chapter 5 and showed what architecture comes into existence. We idealistically use an ontological foundation as meta-meta-model like proposed by Kurtev [29]. We conclude *three type-layers and three meta-layers* are all what is needed to express all information.

The above meta-modelling architecture is not based on any paradigm. In section 6.4 we described how such a paradigm is typically also based on 3 meta-layers. But it is silently also based on formalisms that are part of the top layer and prescribe the instantiation between the lower two layers. Each object-oriented (OO) paradigm variation has such formalisms. BiZZdesign has its own OO paradigm. It is too big a step for BiZZdesign to stop using it. That would need a tremendous effort of which we conclude that it is not affordable for BiZZdesign.

12.2.2 Interoperability

Interoperability results from establishing agreements on communication. Obviously the repository would be interoperable when it complies with a standard for its communication. The Meta Object Facility (MOF) [35] and the Eclipse Modelling Framework (EMF) [55] can both not be used for a BiZZdesign specific repository, as both specify a paradigm that is based on a different variant of object-oriented (OO) than that of BiZZdesign. Moreover both standards do not supply us with scalability solution for the repository's interface.

Compliance to the MOF standard borrows down to working with APIs and interfaces as the standard prescribes. The specification of the MOF [35] pretends models can, with this standard, also form meta-modelling architecture with four or even more meta-layers. Still BiZZdesign's paradigm can not completely be defined when it is used with four layers. The MOF offers no method for defining the formalisms we mention in the previous subsection.

We investigated the EMF and other support from the Eclipse community. The EMF comes with a fixed meta-meta-model, a three-meta-layers architecture and a standard XML-based communication. When its code is used a system representation for models and meta-models is also supplied. In chapter 3 we showed it is not possible to divert from the specific OO variant the EMF is based on: no meta-layer can be added and the top-meta-layer can not be adapted.

In subsection 12.4.2 we will explain standard-based communication is not excluded with our repository design for BiZZdesign.

12.2.3 Scalability

The repository and the tools that use it in fact form a distributed system that follows client-server architecture. In chapter 7 we described that the **practice of data handling** within such a system is domain specific. No investigated technology or framework, including the current technology of BiZZdesign, facilitates the complete freedom of partitioning repository data for any desirable use by any (future) tool.

When the EMF is used the practice of data handling is also domain specific. The EMF does come with a default mechanism for spreading model data over resources, but what data to put in what resource is up to the application. The code generated by the EMF or the extending Graphical Modelling Framework (GMF) [54] just puts all of one model in one resource for instance. Next what is to form one model is entirely up to the user of the generated code.

We developed a scalability solution based on the distributed system. The repository's 'huge amount' of data is left in a database system. The tools in any case handle only a limited subset of this data in their limited memories. This forms their *view* on the data. They should be capable of handling this view and still be able to reach and work with all repository data. The view on the data is dynamically changed by loading and unloading chunks of data we call **clusters**. When to load a cluster and what is to be in a cluster is domain specific; applications load and handle a part of data for a specific use or concern. Hence this should be a part of the definition of a tool that works with the repository. According to our view on meta-modelling in chapter 5 this should be *defined by the meta-model* that defines that tool.

To be able to easily create any cluster from the repository's data, this data must not have any internal borders of any composition. Otherwise the repository's storage would suffer from the tyranny of the dominant decomposition [52]. We need a single, homogeneous storage with elementary units that are as small as possible. This observation had a major impact on our system representation design in chapter 4. There an elementary unit typically contains nothing but an identity. The representation is homogeneous as all relations are considered equal. The original model borders are purely conceptual and are abstracted from.

For scalability the EMF offers a resources system. When an EMF-based application needs to handle its model information partially for scalability reasons it must divide its data over multiple resources. This way it is forced to use *one single, dominant partition* on its data. Obviously this suffers from the tyranny of the dominant decomposition [52]. We found no other ways to obtain scalability when using the EMF non-customized.

Our scalability solution brings new challenges for tools that use the repository. One is how its data should be clustered and moved between the repository and the tool. Another is to allow the tool to modify the data in its view while keeping the repository's data consistent.

12.3 Results

Our work resulted in a **prototype** that is custom and a **design** of a new repository for BiZZdesign. With both we maintained the client-server system architecture from the prior design. Next we incorporated our scalability solution and the data handling and communication between the repository (the server) and the tools (the clients) is based on this solution. We achieve interoperability only by using network-based communication based on a well defined, but custom protocol. We did not achieve it by compliance with a standard or using a (standardized) technology or framework. In the next subsections we describe the design and the prototype respectively and describe what meta-modelling architectures they are based on.

12.3.1 Repository Design

The design of BiZZdesign's repository satisfies all requirements and should not have any of the mentioned problems. It can be used as a storage of any amount of interconnected data any customer of BiZZdesign has that is relevant for describing and modeling its business. It will not necessarily use the *model space* system representation we designed in chapter 4. As alternative to the graph of elements the model space forms, also a graph of objects may be used as system representation.

The repository's meta-modelling architecture is implied by BiZZdesign's current, specific, object-oriented paradigm we mention above. This means the repository of BiZZdesign will just have the three, strict meta-layers and fixed meta-meta-models that correspond to BiZZdesign's meta-model languages. This way the design will exactly suits the needs of BiZZdesign. It is not based on the expressive meta-modelling architecture we described in chapter 6.

With the design we in fact provided a design for a new product in the product line of BiZZdesign. Hence the methodology of software product line engineering [42] applied to the design process. We did not actually develop BiZZdesign's repository. The further, actual development of BiZZdesign's repository will occur after our work. Hence a real proof of our scalability solution is postponed until the completion of this development.

12.3.2 Prototype

We built a prototype solely to prove our scalability solution. As the prototype incorporates our scalability solution, it follows the client-server architecture that is based on and encapsulates a repository and a tool. The communication between the server and the tool is network based. The server represents all data as one model space. The content conforms to our expressive and concise meta-modelling architecture, having three typing-layers and three meta-model layers. Exception is that it contains only a primitive object-orientation model at the top meta-level instead of an ontological foundation. The use of the system representation and this architecture partially proof their capability of representing any desired information.

To obtain a genuine proof facing BiZZdesign it offers a support representative for BiZZdesign's tooling. Like for example the BiZZdesigner tool of BiZZdesign, the prototype tool allows its user to browse a tree of diagrams, open diagrams and close them again. Each action typically leads the prototype tool to load or unload a cluster of elements of its view. Our particular prototype consists of an optimized MySQL [39] database and a repository server and tool client developed using the Java platform. Our prototype diverges from BiZZdesign's repository design on the following points:

- The clusters' definitions are not part of the stored meta-models. They are hard-coded in the tool and repository;
- Data is only read, never modified and written back to the repository;
- No meta-data is read reflectively. In other words only the fixed, not the reflective form of communication we describe in subsection 2.8.5 is used. The tool intrinsically knows all meta-data that corresponds to it;
- The data of the tool mainly contains isolated diagrams. Hence the data is not really as interconnected as with that of the BiZZdesigner, where diagrams can be graphic representations of shared, 'semantic' objects.

12.3.3 Benchmark

Finally we demonstrated scalability by *benchmarking* an original tool of BiZZdesign and the tool of the prototype. We can obviously not guarantee results of test we do with the prototype tool will be the same as with the future repository. Of course we tried to do tests with equal values for as much environment variables as possible. We did tests with both tools on the same workstation. Points of criticism on the benchmark are still:

1. While both tools do work with data that forms a tree of maps whereby the maps contain (reference to) diagrams, they internally represent their data differently;
2. The tools are developed on different platforms. The BiZZdesigner is developed using C++ and the prototype using Java;
3. The prototype server's data does strictly not contain more data than created for the test. The BiZZdesigner tool however creates more (internal) model data to support of functionality it offers, which the prototype does not offer.

The quality of the test was poor. Still the difference in behaviour of the two tools when handling a 'huge' amount of model data was significant. Where the BiZZdesigner gets disproportionately slow (or even crashes), the prototype tool keeps performing well. When we let the prototype handle data that forms a tree of depth 7 containing 167961 diagrams, we observe how the tool opens any node in the tree or diagram within a second. On the other hand the BiZZdesigner stops behaving properly when a model contains more than 2000 similar diagrams, even when no diagrams are even opened yet. From this significant difference we can easily conclude that the data handling approach used in the tools does indeed matter for the scalability.

Of points 1 and 2 of the above criticism we expect they are factors that would make the BiZZdesigner faster then the prototype. Hence they were not paramount. Of point 3 we expect the opposite: The prototype can be faster than the BiZZdesigner as it just loads less data per diagram. Hence it still needs to be contemplated as this can be the factor that points out why the prototype tool is faster than the BiZZdesigner tool. Yet we observe how the prototype tool at least worked with a number of 167961 diagrams and the BiZZdesigner tool with ± 2500 diagrams at maximum. What if we assume the number 167961 was the maximum for the prototype tool to be capable to handle and this factor is the *only* factor that makes the prototype tool faster than the BiZZdesigner tool? That would mean the BiZZdesigner tool has an internal representation of a diagram that is about 167961 / 2000 ≈ 84 times bigger than that of the prototype tool. We think this is implausible. Hence we believe the factor is at least not the only factor making the prototype tool faster than the BiZZdesigner tool. A major part of the improvement in operations with 'huge' data content must originate from our scalability solution.

12.4 Evaluation & Recommendations

We have recommendations for both the use and development of MDE in general and BiZZdesign in particular. For the first created a view and for the second we created solutions.

12.4.1 Model Driven Engineering

We did a broad research to the MDE and we evaluated the MDE in chapter 11. For further development basics of MDE should now be defined. Our model space system representation we described in chapter 4 can be a grounded start. The way a meta-model can define an information system's domain concepts up to the entire system in chapter 5 is based on this. This way all goals of the MDA can be met, on the other hand we criticize the MDA promise. This does not facilitate the impact from a developed information system on its design (its meta-model) as it proposes to automatically derive platform specific details from platform independent ones.

In chapter 6 we continued and placed all relevant models in the models space. We concluded that, as long no paradigm is used, three typing-layers and three meta-layers exactly concise and yet expressive enough. Still a paradigm, like that of BiZZdesign, can not simply be removed. A standard that allows multiple OO and AO paradigms to be described and exist next to each other can once be developed. It would have a four-meta-layer architecture. This

would be beneficial for BiZZdesign and it could achieve interoperability as was once the intention with the MOF.

12.4.2 Solutions

We discourage any use of the technologies, frameworks and standards we investigated for BiZZdesign. Not any allows it to keep its specific object-oriented paradigm. We found the Eclipse community offers much support that would be interesting for BiZZdesign. Still this support is hard to combine with BiZZdesign's current tools. We finally relinquished the use of the EMF for the design of BiZZdesign's repository and the prototype by its scalability problems and the fact it has a strict, different object-oriented paradigm. We recommend BiZZdesign to keep its specific object-oriented paradigm and the corresponding **meta-modelling architecture**.

We did not come up with a repository design that has standard-based **interoperability**. The best thing we did to achieve interoperability is giving a good conceptual base and a clear description and some definitions for the communication: the clustering. On the other hand we do not excluded standard-based interoperability either. Once the repository is set up as designed, it is for example still possible to:

- Load one or more clusters of data from the repository;
- Filter or select from this data some data that conforms to a custom, simplified meta-model. This can for instance be expressed as an ECore model;
- Communicate this data as a standard prescribes. This can for example be using XMI, a Java Metadata Interface (JMI) or an API that conforms to an IDL from the MOF specification.

We recommend BiZZdesign the entire repository design we came up with and the **scalability** solution it incorporated. Our prototype partly proved the feasibility and practice of the solution. We compared it with the current solutions of BiZZdesign and it showed a significant improvement. It would only be really proved when multiple, domain specific, fully developed tools that are used in practice are successfully based on it. We described some challenges with the solution. The prototype tool only reads data. How a tool can modify data while it only has a limited view on the whole of the repository's data and keeping the latter's data consistent is still the most open issue. We think this is feasible and of course the tools that will work with BiZZdesign's repository must be able to modify data within their limited views.

References

[1] Adaptive. (2007). Retrieved 2007, from http://www.adaptive.com/.

[2] Ambler, S. W. (2006). Introduction to Concurrency Control. Retrieved 2008, from http://www.agiledata.org/

[3] Atkinson, C., & Kühne, T. (2002). The Role of Metamodeling in MDA. In proceedings UML 2002 Workshop on Software Model Engineering, pp. 67 – 70, Dresden, Germany.

[4] Atkinson, C., & Kühne, T. (2003). Model-Driven Development: A Metamodeling Foundation. The IEEE Computer Society, volume 20, pp. 36 – 41.

[5] Base64 (n.d.) Retrieved 2008, from http://nl.wikipedia.org/wiki/Base64 Wikipedia, de vrije encyclopedie, http://nl.wikipedia.org/

[6] Berners-Lee, T., Fielding, R., & Masinter, L. (2005). Uniform Resource Identifier (URI): Generic Syntax. Retrieved 2008, from http://tools.ietf.org/html/rfc3986

[7] Bézivin, J., Didonet Del Fabro, M., Jouault, F., & Valduriez, P. (2005). Combining Preoccupations with Models. In Proceedings of the First Workshop on Models and Aspects - Handling Crosscutting Concerns in Model-Driven Software Development (MDSD), ECOOP.

[8] BiZZdesign. (2006). Retrieved 2008, from http://www.bizzdesign.com/

[9] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R, & Grose, T. J. (2003). Eclipse Modelling Framework. Addison Wesley.

[10] Burleson, D. K. (2005). Physical Database Design Using Oracle. CRC Press.

[11] Display a UML Diagram using Draw2D. Retrieved 2008, from http://www.eclipse.org/articles/Article-GEF-Draw2d/GEF-Draw2d.html Eclipse, http://www.eclipse.org/

[12] eBPNM Designer. (2007). Soyatec, Open Solutions Company. Retrieved 2008, from http://www.soyatec.com/ebpmn/

[13] Eertink, H., Belinfante, A., Bakker, h., & Middelink, P. (1998). Testbed Project, D 5.2. Testbed Software Architecture. Internal document of TRC Company.

[14] Eertink, H., Janssen, W., Oude Luttighuis, P., Teeuw, W. B., & Chris A. Vissers. (1999). A Business Process Design Language. World Congress on Formal Methods (1), pp. 76 – 95.

[15] Eichler, H. Scheidgen, M., & Soden, M. (2006). A Meta-Modelling Framework for Modelling Semantics in the Context of Existing Domain Platforms. Humboldt-Universität zu Berlin.

[16] Faase, F. (2008). Design and architecture of the repository. BiZZdesign internal document.

[17] Filman, R. E., & Friedman, D. P. (2000). Aspect-Oriented Programming is Quantification and Obliviousness. In Workshop on Advanced Separation of Concerns, pp. 21 – 35, Addison-Wesley.

[18] Glossary of Networking Terms for Visio IT Professionals. (2002). Retrieved 2008, from http://www.microsoft.com/ Microsoft Corporation

[19] GMF Tutorial Part 2. (2008). Retrieved 2008, from http://wiki.eclipse.org/GMF_Tutorial_Part_2 The Eclipsepedia, http://wiki.eclipse.org/

[20] Gogolla, M., Favre, J., & Büttner, F. (2006). On Squeezing M0, M1, M2, and M3 into a Single Object Diagram. Lecture Notes in Computer Science, Volume 3844/2006, pp. 1 – 9.

[21] Guizzardi, G (2005). Ontological Foundations for Structural Conceptual Models. Telematica Instituut, Fundamental Research Series, vol. 015, Enschede, the Netherlands.

[22] Harel, D., & Rumpe, B. (2004). Meaningful Modeling: what's the semantics of "semantics"? Computer, volume 37, No. 10, pp. 64 – 72.

[23] Hibernate, Red Hat Middleware. (2006). Retrieved 2007, from http://www.hibernate.org/ sourceforge.net.

[24] Iacob, M. (2008). A-MUSE, (Freeband Communication). Retrieved 2008, from http://www.telin.nl/index.cfm?project=A-MUSE

[25] Iacob, M., Jonkers, H., & Wiering, M. (2004). Towards a UML profile for the ArchiMate language. Telematica Instituut & Leiden Institute for Advanced Computer Science.

[26] Kleppe, A., Warmer, J., & Bast, W. (2003). MDA Explained, The Model Driven Architecture: Practice and Promise. Addison-Wesley.

[27] Kühne, T. (2006). Matters of (meta-) modelling. In Software and Systems Modeling (SoSyM), volume 5, number 4, pp. 369 – 385, Springer-Verlag.

[28] Kurtev, I. (2005). Adaptability of Model Transformations. PhD Thesis, University of Twente.

[29] Kurtev, I. (2007). Metamodels: Definitions of Structures or Ontological Commitments? Workshop on Towers of Models, 25 Jun 2007, Zurich, Switzerland, pp. 53-63. University of York.

[30] Lankhorst, M. (ed.) (2005). Enterprise Architecture at Work: Modelling, Communication, and Analysis. Springer.
[31] Lonnee, J. (2007). Ontwerp Repository Metamodel & Portal. Version 1.5, BiZZdesign.
[32] Lonnee, J., & Hoogeveen, S. (2007). Plan van aanpak van invoegen (web)portal. BiZZdesign.
[33] Lozano, F. (1998). Retrieved 2008, from http://www.edm2.com/0612/msql7.html Introduction to Relational Database Design. http://www.edm2.com/
[34] McAffer, J., & Lemieux, J. (2005). Eclipse Rich Client Platform, Designing, Coding, and Packaging Java Applications. The eclipse series, Addison Wesley.
[35] MetaObjectFacility(MOF) Specification. (2002). Object Management Group, Version 1.4. Retrieved 2007, from http://www.omg.org/
[36] Minsky, M.L. (1967). Computation: Finite and Infinite Machines. Prentice Hall.
[37] MOF 2.0/XMI Mapping. (2007). Version 2.1.1, Object Management Group, http://www.omg.org/.
[38] Muth, P. Rakow, T. C., Weikum, G., Brössler, P., & Hasse, C. (1993). Semantic Concurrency Control in Object-Oriented Database Systems. In Proceedings of the Ninth International Conference on Data Engineering, pp. 233 – 242.
[39] MySQL. (2008). Retrieved 2008, from http://www.mysql.com/ The world's most popular open source database. MySQL AB, Sun Microsystems, http://www.sun.com/
[40] OWL Web Ontology Language Guide. (2004). Retrieved 2008, from http://www.w3.org/TR/owl-guide/ W3C, http://www.w3.org/
[41] Plante, F. (2006). Introducing the GMF Runtime. Retrieved 2008, from http://www.eclipse.org/articles/Article-Introducing-GMF/article.html IBM.
[42] Pohl, K., Böckle, G., & van der Linden, F. (2005). Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Heidelberg.
[43] Rashid, A., & Ghitchyan, R. (2003). Persistence as an Aspect. In proceedings of the 2nd international conference on Aspect-oriented software development, pp. 120 – 129.
[44] Richters, M., & Gogolla, M. (1999). A Metamodel for OCL. In UML '99 - the unified modeling language: beyond the standard, pp. 158 – 171.
[45] Rittel, H., & Webber, M. (1973). Dilemmas in a General Theory of Planning. Policy Sciences, Vol. 4, pp. 155 – 169, Elsevier Scientific Publishing Company, Inc., Amsterdam.
[46] Royce, W. W. (1970). Managing the development of large software systems. In proceedings of IEEE WESCON, pp. 1 – 9.
[47] Sheth, A. (1997). Data Semantics: what, where and how? Chapman and Hall.
[48] Sim, S.E., Easterbrook, S., & Holt, R. C. (2003). Using Benchmarking to Advance Research: A Challenge to Software Engineering. In Software Engineering, 25th International Conference on Software Engineering (ICSE'03), pp. 74 – 84.
[49] Stahl, T., & Völter, M. (2006). Model-Driven Software Development, Technology, Engineering, Management. John Wiley & Sons, Ltd.
[50] Stefik, M. J., & Bobrow, D. G. (1986). Object-Oriented Programming: Themes and Variations. In AI Magazine 6(4), pp. 40 – 62.
[51] Tanenbaum, A. S., & Steen, M. van (2002). Distributed Systems, Principles and Paradigms. International Edition, Prentice Hall, Inc.
[52] Tarr, P., Ossher, H., Harrison, W., & Sutton, S. M. (1999). N Degrees of Separation: Multi-Dimensional Separation of Concerns. In International Conference on Software Engineering, pp. 107 – 119.
[53] Testbed. (n.d.). Telematica Instituut. Retrieved 2008, from https://doc.telin.nl/dsweb/Get/Document-2909/Testbed_folder.pdf.
[54] The Eclipse Graphical Modeling Framework (GMF). (2008). Retrieved 2008, from http://www.eclipse.org/gmf/, Eclipse, http://www.eclipse.org/.
[55] The Eclipse Modeling Framework (EMF) Overview. (2006) Retrieved 2008, from http://help.eclipse.org/, Eclipse, http://www.eclipse.org/.
[56] The EMF.Edit Framework Overview. (2004). Retrieved 2008, from http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.emf.doc/references/overview/EMF.Edit.html Eclipse, http://www.eclipse.org/.
[57] The Graphical Editing Framework (GEF). (n.d.). Retrieved 2008, from http://www.eclipse.org/gef/ Eclipse, http://www.eclipse.org/.
[58] Tijhuis, F.P., & Lippe, E. (2004). Handreiking; Kwaliteit Modellen en Bestanden. Retrieved 2008, from http://www.alterra.wur.nl/ Wageningen Universiteit en Researchcentrum http://www.wur.nl/

[59] Weger, M. de (2007). Model Driven Architecture en UML 2. Checklisten Informatiemanagement, 3.C.11.
[60] Wieringa, R.J. (2003). Design methods for reactive systems. Morgan Kaufmann.
[61] XML Path Language (XPath). (1999). Version 1.0, W3C. Retrieved 2008, from http://www.w3.org/TR/xpath
[62] Zhang, Y., & Xu, B. (2004). A Survey of Semantic Description Frameworks for Programming Languages. ACM SIGPLAN Notices, volume 39 no.3, pp. 14 – 30.

Appendix A is intentionally removed. It can be supplied and may be distributed, sold and/or copied only with permission of BiZZdesign B.V. in the Netherlands,

www.bizzdesign.nl