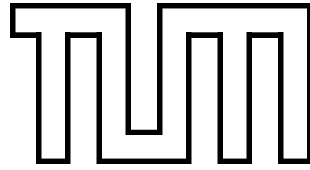
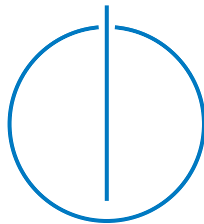


FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masterarbeit in Wirtschaftsinformatik

Tool Support for Federated EA Model Management - An Industrial Case Study

Björn Kirschner



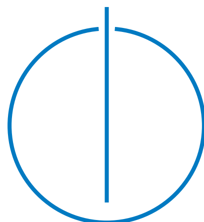
Fakultät für Informatik
Der Technischen Universität München

Masterarbeit in Wirtschaftsinformatik

Tool Support for Federated EA Model Management - An Industrial Case Study

Werkzeugunterstützung für das föderierte Management von Modellen der Unternehmensarchitektur - Eine Fallstudie in der Industrie

Author:	Björn Kirschner
Supervisor:	Prof. Dr. rer. nat. Florian Matthes
Advisor:	M. Sc. Sascha Roth; M. Sc. Marin Zec
Submission Date:	15.05.2014



Declaration

I assure the single handed composition of this master's thesis only supported by declared resources.

München, 15th of May, 2014

Björn Kirschner

Abstract

Enterprise Architecture (EA) management has come to be an important and well-respected discipline, pursuing the vision of aligning business and IT within a company. Yet the documentation of an enterprise's architecture still poses problems, as it is often conducted in a time-consuming and error-prone manual process. The key to automated EA documentation is to utilise existing information maintained by the federated, highly specialised modelling communities. Such a federated EA model management approach can help to tackle common problems related to EA documentation and finally improve EA model quality.

As a holistic design for federated EA model management, this thesis introduces ModelGlue. ModelGlue encompasses concepts for EA model branching, differencing, merging and conflict detection as well as an interactive conflict management dashboard. All these concepts are implemented in a prototypical tool solution.

In order to assess the relevance and aptitude of the proposed concepts, ModelGlue had to be evaluated in a working environment. For this thesis, two case studies were conducted in different companies. Input and suggestions provided by EA practitioners were collected in four (case study A) plus three (case study B) intertwined feedback iterations. Findings from each interview cycle were adapted in the prototype.

Besides the informal feedback obtained in the interviews, data excerpts from EA models of both enterprises were imported into ModelGlue. This data comprised 16 quarterly snapshots of the enterprise architecture with an average of over 18.000 model elements per time-slice. It allowed a profound assessment of ModelGlue's ability to handle realistic data loads.

Contents

List of Figures V

List of Tables VII

1. Introduction 1
1.1. Federated enterprise architecture model management . . . 2
1.2. Introduction of ModelGlue . . . 3
1.3. Research methodology . . . 5
1.4. Outline . . . 9

2. Study design 10
2.1. Research questions . . . 10
2.2. Case and subjects selection and description . . . 11
2.3. Data collection procedures . . . 12
2.4. Validation procedures . . . 13

3. Prototype design 15
3.1. Federated EA model management process . . . 15
3.2. ModelGlue use cases . . . 17
3.3. Introduction of a meta-meta model . . . 19
3.3.1. Role concept . . . 21
3.3.2. Model Conflict Tasks . . . 22
3.3.2.1. Task types . . . 24
3.3.2.2. States of a Model Conflict Task . . . 25
3.3.2.3. Impact of conflict tasks on the state of model elements . . . 26
3.3.2.4. Task assignment . . . 27
3.4. Model branching . . . 28
3.4.1. Branch meta-information . . . 28
3.4.2. Ties between model elements . . . 29
3.4.3. The baseline time . . . 31
3.5. Model merging . . . 33
3.5.1. Operation-based approach . . . 33
3.5.2. Model merging . . . 38
3.5.2.1. Step 1: Element pre-selection . . . 39
3.5.2.2. Step 2: Conflict detection . . . 41
3.5.2.3. Step 3: Conflict classification . . . 41
3.5.2.4. The merge algorithm . . . 42
3.5.3. Conflict classification strategy . . . 45

3.5.4. Conflict resolution strategy . . . 45
3.5.5. Learning and batch-solving . . . 46
3.6. Model differencing . . . 49
3.7. Standard user interface for merge functionality . . . 53
3.8. Visual concepts . . . 56
3.8.1. Conflict resolution dashboard . . . 59
3.8.2. Difference visualisation . . . 60
3.8.3. Graphical interaction . . . 63
3.8.4. Filter functionality . . . 64
3.9. Collaborative conflict resolution . . . 66

4. Evaluation **68**
4.1. Evaluation in an insurance company . . . 68
4.1.1. Initial state of EAM in company A . . . 69
4.1.2. Core meta models of the communities . . . 70
4.1.3. Holistic meta model and abstraction gaps . . . 72
4.1.4. Technical complexity of the models . . . 75
4.1.5. Application scenarios . . . 79
4.1.6. Evaluation of ModelGlue concepts . . . 79
4.1.7. Evaluation of the user interface . . . 83
4.1.7.1. Differencing visualisation . . . 83
4.1.7.2. Merge visualisation . . . 84
4.1.7.3. Conflict table . . . 84
4.1.7.4. Conflict resolution strategy . . . 85
4.2. Evaluation in the health industry . . . 87
4.2.1. Application scenarios . . . 87
4.2.2. Evaluation of ModelGlue concepts . . . 90
4.2.3. Evaluation of ModelGlue visualisations . . . 92
4.2.4. Meta model and impacts on the technical complexity . . . 95
4.3. Comparison of the two cases . . . 98
4.4. Federated EA model management survey . . . 100
4.5. Performance considerations . . . 101
4.5.1. Base version extraction . . . 101
4.5.2. Visual element size calculation . . . 102

5. Related Work **104**

6. Summary **105**
6.1. Conclusions . . . 105
6.2. Limitations . . . 105
6.3. Outlook . . . 106

Bibliography **107**

A. Conflict classification matrices **i**

B. Questionnaire **vii**

List of Figures

1.1. History of related research at the SEBIS chair . . . 2
1.2. Simplified overview of federated EA model management (adapted from [Ro14]) 3
1.3. Overview of the research approach . . . 8

2.1. Intertwined interview sequence . . . 13

3.1. ModelGlue process for supporting federated EA model management [Ro14] . 16
3.2. Use cases for synchronising information sources . . . 18
3.3. Meta-meta model of ModelGlue [KR14] . . . 20
3.4. Mapping of ModelGlue's meta-meta model onto Tricia concepts . . . 21
3.5. Deduction of default roles . . . 23
3.6. Class diagram of the structure around Model Conflict Tasks . . . 24
3.7. States of a Model Conflict Task . . . 27
3.8. States of a Model Element . . . 27
3.9. Original identifier (Oid) evolution of models . . . 29
3.10. Original identifier (Oid) evolution of model elements . . . 30
3.11. Class structure around *ChangeSets* and mapping of *Change* classes to conceptual ModelGlue operations (blue boxes) . . . 34
3.12. ChangeSet and Change in an update example . . . 35
3.13. Overview of the merge algorithm . . . 38
3.14. Data/data part of ModelGlue's strict conflict classification matrix . . . 43
3.15. Simplified sequence diagram of the merge algorithm . . . 43
3.16. Custom conflict resolution rules interface . . . 46
3.17. Learning strategies implemented by Schrade [Sc13] . . . 47
3.18. Dialogue suggesting automated batch-solving of further conflicts, based on a strategy learned by analysing user interactions . . . 47
3.19. Meta model of the information demand for the difference visualization [RM14] 49
3.20. User interface for triggering a model merge process . . . 53
3.21. Interface depicting meta information about a finished merge . . . 53
3.22. Merge result table . . . 54
3.23. Model conflict task details view . . . 55
3.24. Layers of abstraction in a conflict visualisation of the EA model, following De Marco's principle of recursive decomposition [DM78]. Initial implementation by Tobias Schrade [Sc13]. . . . 56
3.25. Schema graph (L1) of the conflict resolution dashboard . . . 57
3.26. Instance list (L2) of the conflict resolution dashboard . . . 58
3.27. Instance neighbourhood graph (L3) of the conflict resolution dashboard . . . 59

3.28. Model conflict task visualisation (L4) of the conflict resolution dashboard . . 60
3.29. Schema graph (L1) in the difference visualisation 61
3.30. Instance list (L2) in the difference visualisation including detailed element information on hover 62
3.31. Instance neighbourhood graph (L3) in the difference visualisation 62
3.32. Three way difference view (L4) as overlay on top of the difference visualisation 63
3.33. Three way difference view (L4) visualising a new instance 63
3.34. Three way difference view (L4) visualising a deleted instance 63
3.35. Graphical interaction functionality provided within visualisations 64
3.36. Filter interface 64
3.37. Recursive JSON filter for one object definition [RM14] 65
3.38. Filtering via MxL 66
3.39. Advanced MxL expression 66
3.40. Collaborative conflict resolution 67

4.1. Application development core meta model 72
4.2. IT asset management core meta model 72
4.3. CMDB core meta model 72
4.4. Abstraction gap resolution user interface presented by [RHM13b] 73
4.5. Meta model 74
4.6. Planned meta model with central application master 75
4.7. Instance neighbourhood graph (layer 3) with 31 instances 78
4.8. Number of core instances between 2009 and 2013 78
4.9. Altered, yellow conflict symbol for indicating ignored conflicts 81
4.10. Potentially misleading background colours in the 3-way comparison visualisation (L4) 84
4.11. Improvement motivated by company A: unambiguous background colouring . 84
4.12. Exemplary application scenario outlined by enterprise architect B 88
4.13. Visual overlays with highly transparent background 93
4.14. Improved overlay background including shadows to emphasise layer separation 93
4.15. Meta model 95
4.16. Part of an exemplary cost allocation graph 97
4.17. Instance list (L2) with 200 instances 98
4.18. Instance list (L2) with 1000 instances 98
4.19. Sequence diagram of extracting the base version of a model element 102
4.20. 42% of the time needed for a model visualisation being spent on placing text elements, 27% on calculating the text height 103
4.21. Comparison of visualisation calculation times with and without caching . . . 103

A.1. Pre-defined strict conflict classification matrix ii
A.2. Pre-defined tolerant conflict classification matrix iii

List of Tables

2.1. Data collection procedures 12

3.1. Mapping of Tricia changes to ModelGlue operations 37
3.2. Conflict classification by Wieland et al. [Wi13]; Source: [Ro14] 42

4.1. Interviews with company A 69
4.2. Interviews with company B 87
4.3. Comparison of prominent aspects of evaluation feedback received from companies A and B 99

A.1. Reasoning behind the strict conflict classification matrix (part 1) iv
A.2. Reasoning behind the strict conflict classification matrix (part 2) v
A.3. Reasoning behind the strict conflict classification matrix (part 3) vi

1. Introduction

In most modern enterprises, information technology (IT) complexity has increased significantly throughout the last years. Apart from its original function as mere support for the company's business, IT has recently shifted towards being seen as an 'enabler' for existing and new business models. Considering these changing circumstances, IT management has become a crucial factor for the success of enterprises [WR09]. In this context Enterprise Architecture (EA) management strives to align viewpoints of business and IT and provide a holistic view of the enterprise [La05].

As information about a company's enterprise architecture often is of extensive scope and complexity, considerable potentials lie in the automation of EA documentation. Despite a variety of EA tools on the market (see e.g. [Ma08b] and [Be12] for an overview), EA documentation still requires substantial manual efforts. Surveys by Winter et al. [Wi10] and recently by Roth et al. [Ro13] showed that only a small number of companies already apply automated approaches for EA information collection.

Challenges on the path towards automated EA architecture documentation have been investigated in various publications of the SEBIS chair (software engineering for business information system) at TU München. Figure 1.1 shows an excerpt of recent work on this topic.

Buschle et al. [Bu12] investigated how to alleviate the time consuming and error-prone process of EA documentation via extracting information out of a productive system. In particular, they evaluated to which degree of coverage productive data is appropriate for EA models. Based on Buschle's work, Grunow et al. [GMR12] analysed data quality aspects necessary for automated EA documentation.

Hauder et al. [HMR12] conducted a survey about challenges for automated EA documentation. Amongst other findings, they identified the abstraction gap[1] between EA and information sources as being a major obstacle for automation ambitions. Farwick et al. [Fa13a] reports on key problems in EA documentation. They focus on the appropriateness of information sources for automated EA documentation.

[1]Abstraction gaps are also explained in section 4.1.3

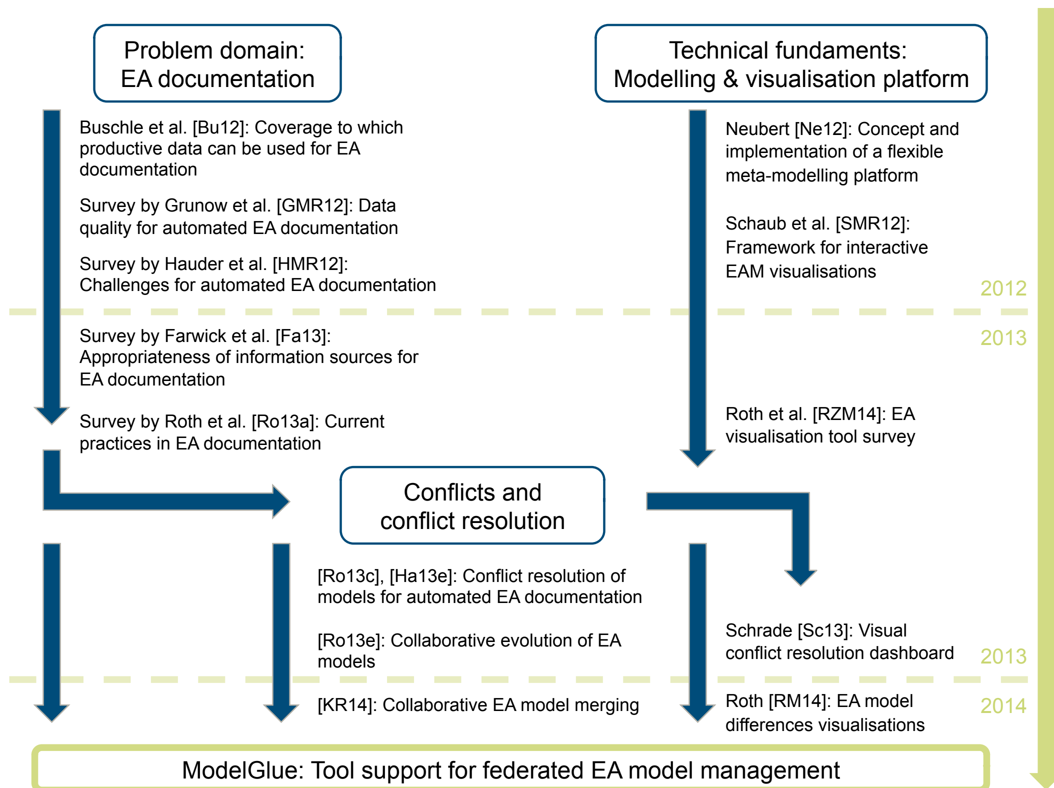


Figure 1.1.: History of related research at the SEBIS chair

Roth et al. [Ro13] found out that manual data gathering is still the prevailing collection method within a vast majority of companies and only about a third of EA practitioners apply partially automatic collection approaches. Additionally, they identified (amongst other factors) the following three primary challenges in EA model maintenance:

- Huge data collection effort
- Low EA model data quality
- Insufficient tool support

1.1. Federated enterprise architecture model management

In order to approach the challenges of automated EA documentation described in the previous section, the SEBIS chair adopted the concept of **federated enterprise architecture model management**, originally introduced by Fischer, Aier and Winter [FAW07].

Federated EA model management assumes that companies are are structured into different modelling communities. Each of these communities deals with its own models and maintains productive data in its own highly specialised IT systems. However, as they are part of a federation, models of the communities often overlap in certain aspects. In order to enable

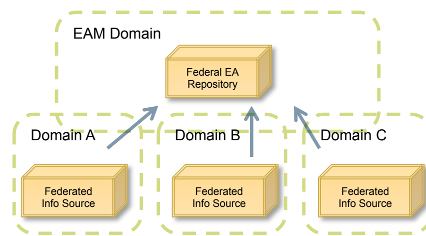


Figure 1.2.: Simplified overview of federated EA model management (adapted from [Ro14])

a holistic view on the entire enterprise architecture of a company, data and models from all relevant communities have to be integrated into a central EA repository (see figure 1.2).

One of the challenges of this integration of federated information sources are the mentioned model and data overlaps between different modelling communities. Due to these overlaps, even an automated integration process might come across situations where it faces conceptual problems between different models which cannot be solved by an algorithm. If this is the case, manual conflict resolution has to be conducted. In [RHM13a], Roth et al. suggest means to solve model conflict tasks collaboratively, involving EA stakeholders and experts.

Roth, Hauder et al. [RHM13b, Ha13b] provide concepts tackling the problem of how model conflicts can be addressed to appropriate receivers. They also implemented a prototypical solution supporting the resolution of model conflicts and abstraction gaps. Based on these foundations, Kirschner and Roth [KR14] implemented further functionality for federated EA model management, with a focus on model conflict detection and the dissemination of model conflict tasks (cf. section 3.3.2).

1.2. Introduction of ModelGlue

Based on the vision to tackle common problems of automated EA documentation (described at the beginning of this chapter) via a tool, **ModelGlue** was born. ModelGlue intends to provide comprehensive tool support throughout all stages of federated EA model management.

On a technical level, ModelGlue's foundations reach back to a platform implemented by Büchner [Bü07], nowadays called 'Tricia' [in14]. The basis for flexible modelling of (enterprise architecture) information was laid by Neubert [Ne12], who introduced the concept of 'hybrid wikis'. Information about the visualisation framework used by ModelGlue can be found in Schaub et al. [SMR12].

ModelGlue itself was implemented by Sascha Roth, Tobias Schrade and Björn Kirschner. Schrade contributed an interactive conflict resolution dashboard (see section 3.8.1), implemented for his bachelor's thesis [Sc13]. Details on EA model merging and conflict detection are reported in [KR14]. Aspects of the construction of model differences visualisations can be found in [RM14].

This thesis, as well as Roth's PhD thesis [Ro14], try to provide a detailed account of the holistic conception of ModelGlue as well as its design and implementation.

Since in summer 2013 the evolution of ModelGlue proceeded quickly and more and more functionality was contributed to its implementation, finally an evaluation of its concepts became necessary in order to confirm its ability to cope with realistic scenarios.

1.3. Research methodology

Norwegian researchers around Sjøberg formulated the vision that empirical research in the Software Engineering (SE) domain shall come to "guide the development of new SE technology" as well as influence important SE decisions in industry [SDJ07].

Upon the path of such sublime ambition, this work seeks to conduct and report a profound and comprehensive evaluation of concepts and implementation of ModelGlue, a tool facilitating federated EA model management. With Sjøberg's vision in mind, findings presented here might come to be considered by future researchers advancing the same or related problem domains as well as by industry practitioners tackling the challenge of efficient EA model management.

As laid out in the previous chapter, motivation for putting effort into researching solutions for EA model management is already well documented in scientific literature. Several surveys [Be12, Ro13] aimed at a thorough understanding of current problems and industry needs in the EAM domain. From the perspective of this work, these explorative surveys [Bo90] can be seen as pre-studies specifying problems relevant in the industry, preparing for deeper investigation (thus following research recommendations by Wohlin et al. [WHH03]).

Based on these revelations and problem specifications, the Sebis chair started to delve into the topic and develop solutions for federated EA model management. The outcome, a set of concepts as well as a prototypical implementation called ModelGlue, seeks to alleviate some of the problems identified in industry. Two conference papers [RM14, KR14] already report on some conceptual and technical insights into ModelGlue.

Consequently, ModelGlue is engineered by a university chair without closer involvement of external companies in early development stages, which arguably might lead to a researcher bias.

To address this potential scientific bias, align ModelGlue to industry needs and refine the proposed solutions under real-world circumstances, the subsequent step presented within this work is an evaluation of ModelGlue.

Positioning along the dimensions of empirical research

Empirical research can be categorised along various different dimensions. In the following, the most important dimensions will briefly be explained in order to facilitate the positioning of this work.

Purpose: Firstly, Robson [Ro02] classifies four purposes of research methodologies:

- Exploratory: searching for new insights and hypotheses.
- Descriptive: illustrating a phenomenon.
- Explanatory: finding explanations for known problems.
- Improving: seeking to improve certain aspects of a phenomenon.

Purpose of this work is particularly of improving nature. Most of the exploratory part was done in the many pre-studies portrayed in the previous section. Robson's classification will not be considered as strict boundaries within this work, so elements of each purpose can be found. But the focus lies on the improvement of ModelGlue considering realistic circumstances.

Data collection: Empirical research can be conducted either qualitatively or quantitatively. This distinction, however, does not need to be seen exclusively. Seaman argues that combining qualitative and quantitative data collection often facilitates the understanding process and thus improves research results [Se99].

Throughout many parts, this thesis relies on qualitative data collection, but as it is a technical work, often with a strong tendency towards quantitative measures, especially in the final validation part.

Scope: Basili et al. distinguish between single-project studies, multi-project studies, replicated project studies and blocked subject-project studies [BSH86].

As this work conducts and examines a study in two different enterprises, it can be considered a replicated-project study.

Triangulation: Triangulation describes the approach of looking at the study from various different angles. This can enhance the precision of the research results, especially when handling with qualitative data.

Triangulation can be divided into the following four types [St95]:

- *Data (source) triangulation:* Data collected from various sources.
- *Observer triangulation:* More than one observer.
- *Methodological triangulation:* Application of different data collection methods.
- *Theory triangulation:* Alternative theories or viewpoints.

This thesis strives to follow the first three of these recommendations. Data triangulation is ensured by conducting case studies in two different enterprises. At every interview, at least a second researcher was present (observer triangulation). The final survey appends quantitative aspects to the mostly qualitative case studies (methodological triangulation). Different viewpoints are illustrated by two fellow researchers: Pouya Aleatrati Khosroshahi, who investigated governance and process structures of federated EA model management in his Master's thesis [Kh14] and Sascha Roth, who advanced the topic at a broader scope in his PhD thesis [Ro14].

Methodology: Because of its high abstraction level, the subsequent sub-sections follow Wohlin's division between the following four methodologies for evaluating software: **experiments, case studies, surveys, and post-mortem analysis[2]** [WHH03]. The sections

[2] A post-mortem analysis is a retrospective examination of a usually already finished project [WHH03]. Since the idea of this methodology is inapt for evaluating a prototypical implementation like ModelGlue and conceptual ideas in an early development phase, post-mortem analysis will not be covered in an own chapter.

explain the applicability of each methodology under the circumstances of this project and justify the set chosen for this work.

Experiment

Experiments are usually conducted in a controlled laboratory environment. They measure the effects of one variable while manipulating another variable [Ro02]. Their control at precisely manipulating certain variables and record consequences makes them suitable especially for comparisons [Mo97]. They cover a limited scope and are thus often referred to as "research-in-the-small" [KPP95].

Because of its limited scope, experimentation cannot be the methodology of choice for evaluating the holistic approach of ModelGlue and the entirety of its concepts. What is more, controlled experimentation usually is a time-consuming process few real-world EA practitioners would be willing or able to participate in.

Case study

Case studies, like most empirical research methods, have their roots in social sciences. Investigating the applicability of case studies for software engineering, Runeson and Höst summarize the definitions of Robson [Ro02], Yin [Yi03] and Benbasat et al. [BGM87] as follows: A "case study is an empirical method aimed at investigating contemporary phenomena in their context" [RH09].

Because of the realistic set-up, Kitchenham et al. introduced the term "research-in-the-typical", as examining subjects interacting in their typical application context is the major goal of a case study [KPP95]. Case studies do not offer the means to retrieve as much detailed information about causal relationships as controlled experiments do, but instead focus on deeper understanding of certain phenomena under investigation. Level of control and degree of realism are contradicting characteristics, so the right trade-off between the two has to be found [RH09].

In the scientific community there is still dissent about whether or not case studies can be generalised from. While many design science researchers like ...[] and Kitchenham et al. find it difficult to generalise results obtained in a case study [KPP95], the trend goes to a wider acceptance of case studies, reasoning that "knowledge is more than statistical significance" [RH09, Le89]. Flyvbjerg emphasises the "force of example" and states that even a single, well-conducted case study can "contribute to scientific development"[Fl06]. His detailed argumentation can be found in [Fl01].

To address remaining doubts about generalisability of a case study, this thesis bases upon two intertwining cases and validates core findings in a final survey providing quantitative data.

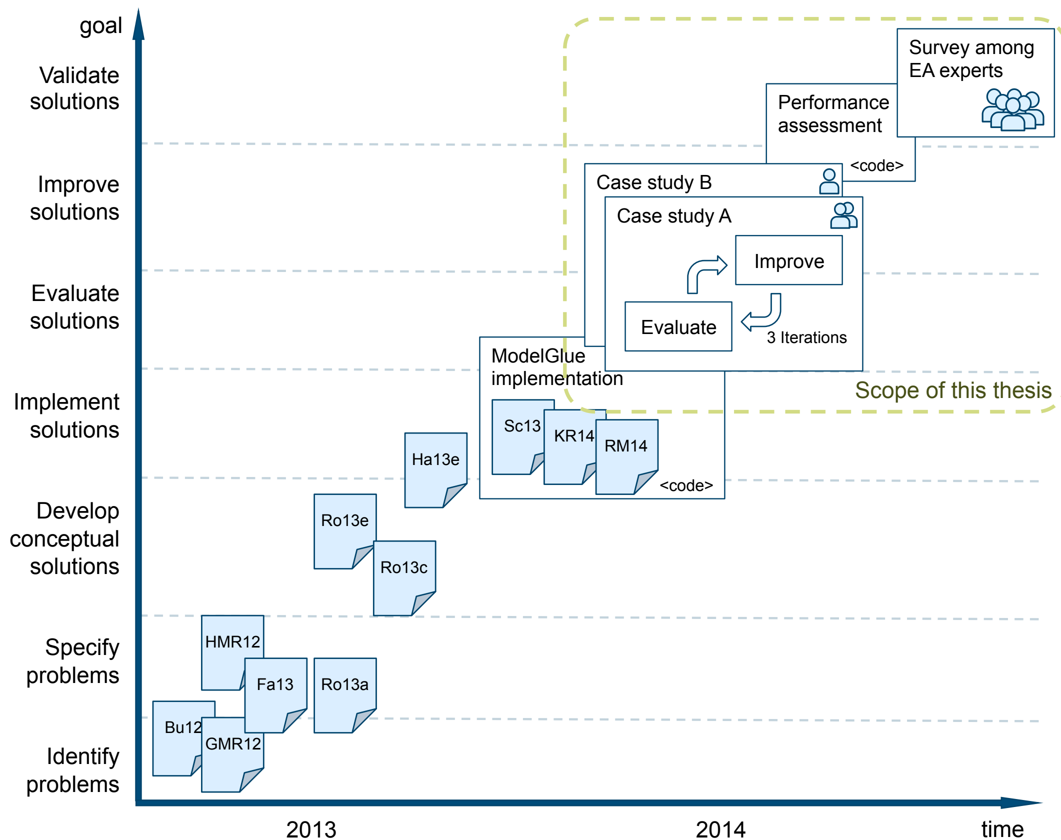


Figure 1.3.: Overview of the research approach

Survey

Robson defines surveys as a "collection of standardised information from a specific population, or some sample from one" [Ro02]. As it is the quickest way to address large populations, surveys are usually conducted via questionnaires.

Summarising the research approach

Figure 1.3 summarises the research approach of this thesis in the context of preceding work conducted at the SEBIS chair. Visualisation of this process was inspired by Shull et al. [SCT01].

Our main goal was to gain insights about how ModelGlue copes with realistic circumstances. Therefore, two case studies are presented in this thesis. Both studies were laid out in an iterative manner, so that feedback could be adapted and implemented iteratively. A performance assessment of real-world data was performed in order to assess ModelGlue's ability to cope with realistic data loads. Finally, an online survey shall assess general validity of the feedback received in the case studies.

1.4. Outline

The remainder of this thesis complies with the following structure: At first, chapter 3 offers a detailed report on the design of ModelGlue, its process, concepts, algorithms and user interface. Then chapter 4 describes findings obtained in the evaluation phase. After a short discussion of related work in section 5, chapter 6 finally summarises important results.

2. Study design

This chapter starts with the declaration of the research questions of this thesis. Thereafter, section 2.2 details the subjects involved in our case studies. Section 2.3 justifies data collection procedures applied in these case studies. Finally, section 2.4 discusses validation procedures.

2.1. Research questions

This thesis strives to answer the following research questions:

Q_1: Does the concept of federated EA model management reflect industry needs?

Q_{1_a}: Have all relevant use cases been identified?

Q_{1_b}: Are there other potential application scenarios for this concept?

Enterprise architecture is, as the name suggests, a concept applied almost entirely in organisations - particularly in large enterprises where the additional effort for EAM helps to manage the complexity of IT systems. Consequently, also our approach of federated EA model management aims at larger companies. Therefore, it has to reflect real-world industry needs and fulfil all relevant use cases in order to be accepted by users.

Q_2: How does ModelGlue support EA practitioners who seek to automate EA model maintenance?

If federated EA model management is generally appreciated, how can ModelGlue's approach towards its concepts help EA practitioners with regard to our primary vision - the automation of substantial parts of EA model maintenance.

Q_3: What are technical industry constraints and how could they affect ModelGlue's design?

Q_{3_a}: How frequently are information sources synchronised with the EA repository?

Q_{3_b}: What amount of data is relevant in EA models? How complex are typical EA models?

Q_{3_c}: Does ModelGlue scale considering realistic data loads?

In order to align the ModelGlue prototype to realistic application circumstances, we want to identify potential technical constraints and their impact on ModelGlue's design. Relevant technical factors which have to be considered in this regard include the application frequency and EA model and data size and complexity. Knowing these aspects allows an assessment of whether ModelGlue scales under realistic circumstances.

> $\boldsymbol{Q_4}$: Does the implementation of ModelGlue (behaviour, UI, ...) meet user expectations?

Finally, we aim for elaborate user feedback on our implementation of ModelGlue. Does it behave as expected? Are user interfaces intuitively comprehensible? What can be improved in our solution?

2.2. Case and subjects selection and description

The case studies described in this thesis were conducted in cooperation with industry partners of the SEBIS (software engineering for business information systems) chair at TU München. Both company A and company B had cooperated with the chair before. All interview partners participated out of intrinsic motivation and received no compensation or benefits for their efforts. Therefore we would like to thank them for the elaborate feedback given in various occasions. All dialogue partners work in the field of EAM.

Company A: This case study took place in a large German insurance company. With a revenue of a few billion euros and nearly ten thousand employees, it serves more than ten million customers[3].

Partners in company A were two enterprise architects, denoted by the abbreviations A_1 and A_2 throughout this thesis.

Company B: Company B is a medium-size organisation in the health industry. It employs roughly 1500 employees, 150 thereof in the IT department, and represents the interests of several ten thousand members.

Contact person here was enterprise architect B, currently the only employee engaged full-time in EAM in company B.

Company C and D: In order to gain verifying insights from additional viewpoints, two further interviews were conducted. One in company C, an IT consulting enterprise with several hundred employees and a focus on supporting financial service providers. The other in company D, a global technology consulting company with several ten thousand employees.

[3] All figures from 2012

Table 2.1.: Data collection procedures

Month	Company	Type	Structure
Sept. ’13	A	Archival data	-
Sept. ’13	A	Interview (phone)	Unstructured
Dec. ’13	B	Interview (personal)	Semi-structured
Dec. ’13	B	Archival data	-
Dec. ’13	A	Interview (personal)	Semi-structured
Jan. ’14	C	Interview (personal)	Semi-structured
Jan. ’14	B	Interview (personal)	Semi-structured
Feb. ’14	A	Interview (phone)	Semi-structured
Mar. ’14	D	Interview (personal)	Unstructured
Apr. ’14	-	Questionnaire	Fully structured

2.3. Data collection procedures

Following the principle of data source triangulation [St95] described in section 1.3, data presented in this work was collected from various different data sources. Effects of drawing biased interpretations based on only one single source could thus be diminished [RH09].

Table 2.1 gives an overview of how data was collected in all study iterations. In conformity with Kitchenham et al. [Ki02], this table answers the questions ‘who’, ‘when’ and ‘how’.

Case A: Data collection took place in four iterations:

After the agreement on mutual cooperation, company A provided exports from all information sources relevant for their EA in the years 2009 to 2013. Lethbridge et al. would call such an analysis of already available data a *third degree* data collection technique, compared to *first degree* (direct methods, e.g. interviews) and *second degree* (indirect methods, e.g. usage behaviour recording) techniques [LSS05]. Third degree data collection is usually cheaper than first and second degree methods, but offers less control to the researcher [RH09]. Since the provided exports had been dedicated to company A’s EA team, the concern of van Solingen and Berghout [VSB99] that third degree data has usually been recorded for a different purpose than research projects can be allayed.

In the first 80 minute phone call with one of the enterprise architects from company A (A_1) he revealed qualitative insights into how company A advances EAM, explained their EA model and problems they face in this domain. Classifying it into Robson’s categories (unstructured, semi-structured and fully structured [Ro02]), this was an unstructured interview pursuing exploratory objectives.

Iteration three was a three-hour interview with two enterprise architects from company A (A_1 and A_2). It took place in the the headquarters of company A. The meeting was conducted

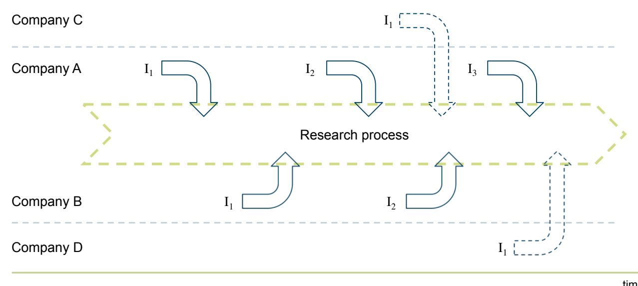


Figure 2.1.: Intertwined interview sequence

in a semi-structured manner, i.e. the prepared questions (similar to the questionnaire in appendix B) were taken as a non-committal guide throughout the session. This approach allows to promptly and adequately respond to new ideas [Ro02]. Questions were asked in a *funnel* structure [RH09], beginning with general questions about the company and EA processes and ending with specific questions asking for an assessment of the implementation of ModelGlue.

Finally, a two-hour phone call with architect A_1 formed another semi-structured interview.

Case B: This case study started with a two hour semi-structured interview with the enterprise architect B at the headquarters of company B.

The next step was company B providing the chair with EA data for further analysis. In contrast to the data provided by company A, this data did not comprise historical information but only parts of the current EA model.

In a second two hour meeting at TU Munich, B was presented the ModelGlue implementation. This interview was also conducted in a semi-structured manner.

Interviews with **companies C and D** served validity purposes and are thus relevant for the research process, but will not be detailed in this thesis. Further information about these two occasions can be found in [Kh14] and [Ro14].

2.4. Validation procedures

Yin divides validation into four aspects: construct validity, internal validity, external validity and reliability [Yi03].

External validity, i.e. whether research findings can be generalised, was taken care of by an intertwined interview sequence illustrated in figure 2.1. After each interview with company A we conducted an interview with company B and vice versa. Thus, we had several iterations where opinions and suggestions of one party could be validated by the other party.

Final validation of our research results and proposed scientific solutions was done in an online survey.

Reliability refers to the question whether another researcher would draw the same conclusions out of the same case. To address such a potential researcher bias, all interviews were performed and analysed by two and sometimes three researchers. Sascha Roth, who was present at all occasions, describes his interpretations in his PhD thesis [Ro14]. Pouya Aleatrati Khosroshahi, who participated in some of the interviews and conducted several more, documented his findings on the federated EA model management approach in his master‘s thesis [Kh14]. He advanced the topic from a slightly different viewpoint as he investigated governance and process structures.

3. Prototype design

This chapter describes important facets of ModelGlue concepts and the design of a prototypical implementation. It starts with the fundamental process we support for federated enterprise architecture model maintenance, followed by an overview of use cases supported by ModelGlue's implementation. Thereafter, static aspects documenting the software architecture (section 3.3) are laid out. This section finishes with an emphasis on behavioural and interaction facets (from section 3.4 onwards). The latter part includes detailed analysis of ModelGlue's major application scenarios *branch*, *merge* and *show differences*.

3.1. Federated EA model management process

ModelGlue's vision is to support federated EA model management throughout its entire process, embracing all relevant tasks and providing solutions for typical challenges. Figure 3.1 shows a comprehensive process developed by Roth [Ro14] in BPMN notation [Ob11a]. This federated EA model management process may be seen as a thread guiding through ModelGlue's functionality.

Integrate information source. Federated EA model management starts when the EAM team identifies the necessity to integrate a new information source into the EA model. Main goal of this step is to conceptually align the new information source with the already existing EA model. This includes establishing a common understanding of model concepts and terminology.
A typical problem of this phase could be that community A modelled the type 'Service' from a business level perspective, meaning a service offered to customers of the enterprise. Community B maintains models of the company's IT infrastructure. As B follows the philosophy of a 'service-oriented architecture' (SOA), their model also comprises a type 'Service', but in a context totally different from the 'Service' modelled by community A. Before integrating the two information sources from communities A and B, the EA team has to establish a solution for the semantical mismatch of the two types called 'Service'. In this case the semantical mismatch might be solved by simply renaming one to 'Business Service' and the other to 'IT Service' and thus separating the two concepts. In other cases the conceptual alignment could be more complicated and involve changing a considerable part of the models which are meant to be integrated. Section 4.1.3 details this process in the context of company A. It also explains the concept of *abstraction gaps* as well as an existing solution implemented by Roth et al. [RHM13b]. Further detailed information as well as sub-processes of this step can be found in Roth's work [Ro14].

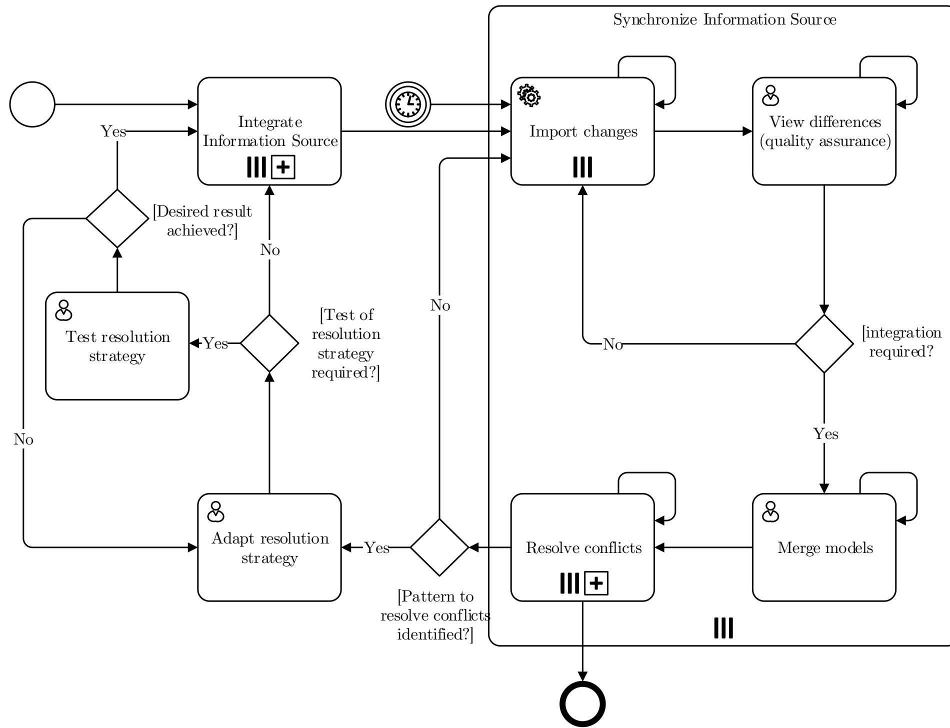


Figure 3.1.: ModelGlue process for supporting federated EA model management [Ro14]

The process of integrating information sources can prove to be rather tedious. In case of company A, several months were needed to conclude this step.

Synchronise information source: Import changes. After an information source has been conceptually integrated, its contents can be synchronised. Central starting point therein is the import of new changes. This step may be triggered manually by the EA team or via a scheduled routine, e.g. every month. Import changes is a rather technical step which deals with the transfer of data (or preferably only changes - see section 3.5.1 for more detailed information about this operation-based approach) from the original information source into the EA repository. This step usually requires a well-defined interface for the translation of data between sources and the EA tool. If most imports are triggered manually, even csv-files (comma separated values) might serve as a light-weight solution. See section 4.2.2 for further discussion on data import expectations.

View differences. After changes of an information source have been imported, the EA team can review these alterations. Via differencing capabilities (described in section 3.6) they can determine whether the imported changes are relevant for the EA model. Viewing differences also facilitates checking correctness of the changes and the resulting model. If the change import led to obvious technical errors or inconsistent data, this would be the time when the EA team could decide to discard this import, solve potential errors in the source information system or the import process and eventually try another import.

Merge models. If the EA team could not find any obvious technical or conceptual problems that would impede further actions, the next step would be to merge the newly updated model into the holistic EA model. Merging is carried out by an algorithm that detects potential conflicts and consolidates non-conflicting parts of the given models. See section 3.5.2 for a detailed report on this topic.

Resolve conflicts. Merging models - in EAM like in other disciplines - often leads to conflicts. These conflicts, detected by the merge algorithm, can only partly be solved by automated routines or heuristics (see section 3.5.4 for ModelGlue's automated resolution capabilities). In many cases manual user interaction is still necessary to resolve them. Amongst others, interesting questions in this regard are: How can conflicts be assigned to appropriate recipients? And how can users be supported in the resolution process?
ModelGlue uses special tasks for conflict representation (see section 3.3.2) and provides sophisticated visual functionality for their resolution (section 3.8).

Adapt and test resolution strategy. A merge resulting in a multitude of similar conflicts might be an indication of missed potentials in the automatic conflict resolution strategy. If that is the case, ModelGlue allows users to define or refine custom conflict resolution rules (see section 3.5.4). After testing these rules they will be considered and applied by the merge algorithm for the next synchronisation step.

3.2. ModelGlue use cases

The process introduced in the previous chapter helps to infer use cases which should be supported by a tool for federated EA model management. Figure 3.2 shows selected use cases of the process step 'synchronise information source'. The process step 'import changes' is not listed as it is seen as a technical prerequisite for all other use cases. Actors have deliberately been omitted in the diagram as most use cases will be carried out by an enterprise architect. For a detailed evaluation of roles and responsibilities within the process of federated EA model management Aleatrati's work [Kh14] can be recommended. Use cases related to conflict resolution are partly based on Schrade's thesis [Sc13].

Compare models. As we consider models to consist of a schema and a data part, models may be compared on different abstraction levels. On both levels, viewing differences of relations between model elements is of vital importance. Differencing functionality is described in section 3.6.

Merge models. In a first step, merging models allows to specify certain parameters and preferences which influence how the merge is conducted. As a central point, specifying the resolution strategy can be named. A user may choose whether to apply a strict or a tolerant conflict detection regime. Additionally, custom conflict resolution rules can be specified

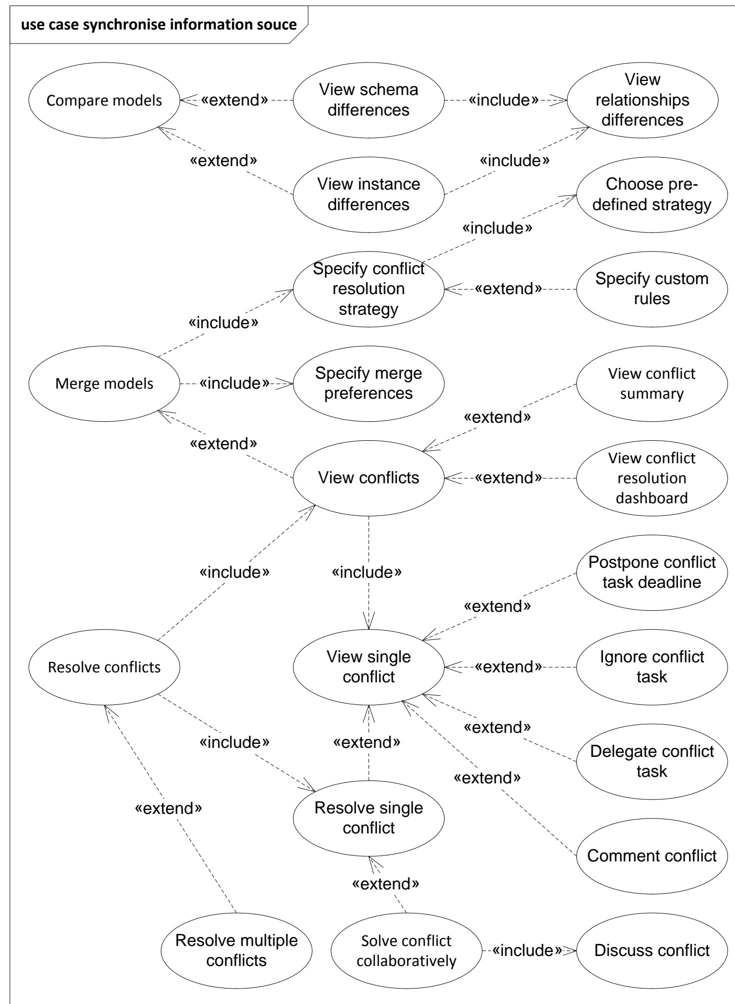


Figure 3.2.: Use cases for synchronising information sources

which foster automated conflict resolution. After a merge, resulting conflicts are presented to the user either in a compressed summary or in an advanced visual conflict resolution dashboard. Merging functionality is explained in section 3.5.2, details on visualisations in section 3.8.

Resolve conflicts. Merge conflicts in ModelGlue are always wrapped in a 'model conflict task' (see section 3.3.2). Upon viewing, such a task can be postponed, ignored[4], delegated or commented.When resolving a conflict, users are always given full control over the corresponding task, i.e. they can perform all task-related functionality mentioned before. Another use case is to solve conflicts collaboratively, where multiple users can discuss and perform actions directly inside the tool (see section 3.9).

[4]Note that 'ignore conflict' is the adoption of a suggestion raised by company A in the case study (see section 4.1.6).

3.3. Introduction of a meta-meta model

Figure 3.3 shows the conceptual meta-meta model ModelGlue is based on (introduced in [KR14]). The terminology used for model descriptions throughout this thesis is based on Atkinson and Kühne's [AK03] endeavours toward a logical conception of model level abstractions. Based on the 'Meta Object Facility (MOF)' standard defined by the 'Object Management Group' [Ob14], they added an ontological meta modelling view. In strict MOF syntax, ModelGlue's meta-meta model comprises constructs of M1 (data) as well as M2 (schema/meta model) level.

A *Model* consists of *ModelElements*. *Objects*, *Attributes* and *Values* constitute the **data part of the model**, whereas *ObjectDefinitions* and *AttributeDefinitions* are the **schema part** which may be seen as a meta model of the data part.

Via the 'parent'-composition, *Objects* are hierarchically organised in a tree data structure. Every *Model* comprises exactly one *Object* tree with the 'rootItem' being the root node of the tree. *Objects* also comprise a set of *Attributes*. Apart from a name, *Attributes* consist of a list of *Values*. This structure allows an *Attribute* to contain *Values* of different types. *Attributes* may conform to an *AttributeDefinition* which specifies multiplicity and permitted value types. If an *Attribute* does not conform to any definition, it is called 'free attribute'. Such freedom fosters flexibility in an early modelling phase [MNS11] and a bottom-up modelling approach [Ne12]. *Objects* may conform to an *ObjectDefinition* (also called 'type'), which aggregates *AttributeDefinitions* and thus specifies which *Attributes* the *Object* is obliged to implement. Throughout this work the term 'is of type' will be used as a synonym to 'conforms to'.

Relationships between *Objects* are modelled via *Attributes* enclosing *LinkValues*. Associations between *ObjectDefinitions* are modelled by restricting an *AttributeDefinition* to *LinkValues* leading to one specific other *ObjectDefinition*.

Example: The schema of an Enterprise Architecture contains an *ObjectDefinition* called 'Application', specifying the two *AttributeDefinitions* 'Availability' and 'Users'. 'Availability' requires exactly one *Number* value (this is a *Constraint*). 'Users' allows a list of *LinkValues* which may only contain relations to *Objects* of the type 'ApplicationUser' (another *Constraint*). One *Object* implementing the type 'Application' might be called 'jBoss 7.1.1', specifying the *Attribute* 'Availability' as required by the definition via the *Value* '95'. Additionally, this application is associated to an element 'user142' which complies to the *ObjectDefinition* 'ApplicationUser'.

Technical environment: ModelGlue is implemented in **Tricia**, a software platform for collaborative project and information management [in14]. Visualisations in Tricia are based on a plug-in developed at the SEBIS chair at TUM. The Tricia GraphicalVisualizations plug-in is the technical successor of SyCaTool [Sc06], a solution developed especially for visualising system maps [Wi07]. Conceptual foundations of the GraphicalVisualizations plug-in are described in [SMR12]. On implementation level, many concepts used throughout this thesis are realised via standard Tricia concepts. Figure 3.4 shows fragments of the Tricia

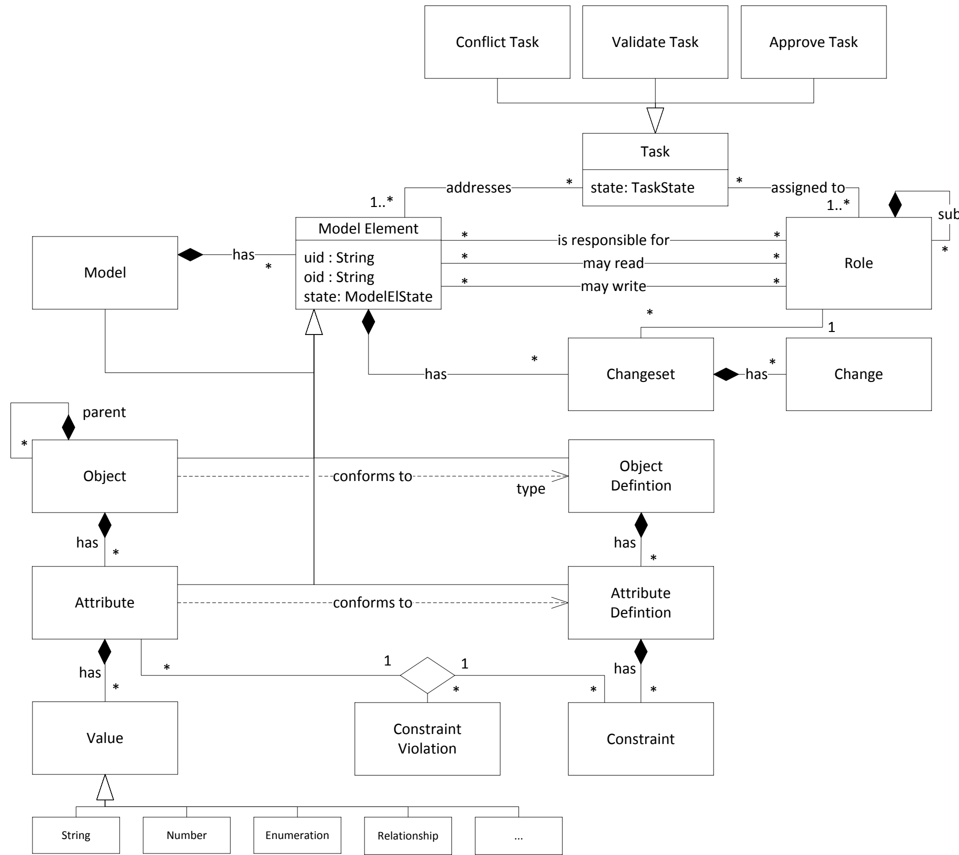


Figure 3.3.: Meta-meta model of ModelGlue [KR14]

class structure and an important part of the terminology mapping between ModelGlue's conceptual meta-meta model (see figure 3.3) and Tricia's meta-meta model.

Mapping of ModelGlue's meta-meta model onto Tricia concepts: Most importantly, what we refer to as *Object* is realised via *Pages* in Tricia. The part of a page containing structured information is called *Hybrid*, based on the concept of 'hybrid wikis' developed by Neubert [Ne12]. *Page* and *Hybrid* are linked via the *Mixin* class, a concept which allows simple functionality amendments and reuse similar to multiple inheritance. *Object-Definitions* are called *TypeDefinitions* in the implementation. The set of *Pages* and *TypeDefinitions* forming a *Model* is realised via *Spaces* in Tricia. *Attributes* are implemented as *Properties* in Tricia. The implementation distinguishes a multitude of different types of properties (see [Bü07] for all property subclasses). Every property type has its own *Value* type, e.g. a *StringProperty* contains *StringValues*.
Implementing ModelGlue's role concept posed a challenge. Originally, only *PageSpace* and *Page* had right roles implemented directly in their classes. In order to use read, write and responsible roles in all other model elements, too, the respective functionality was shifted into a mixin called *Rights*. This mixin can be used in all *PersistentEntities*, consequently also in the classes *TypeDefinition* and *PropertyDefinition*. As *Properties* are *Features*

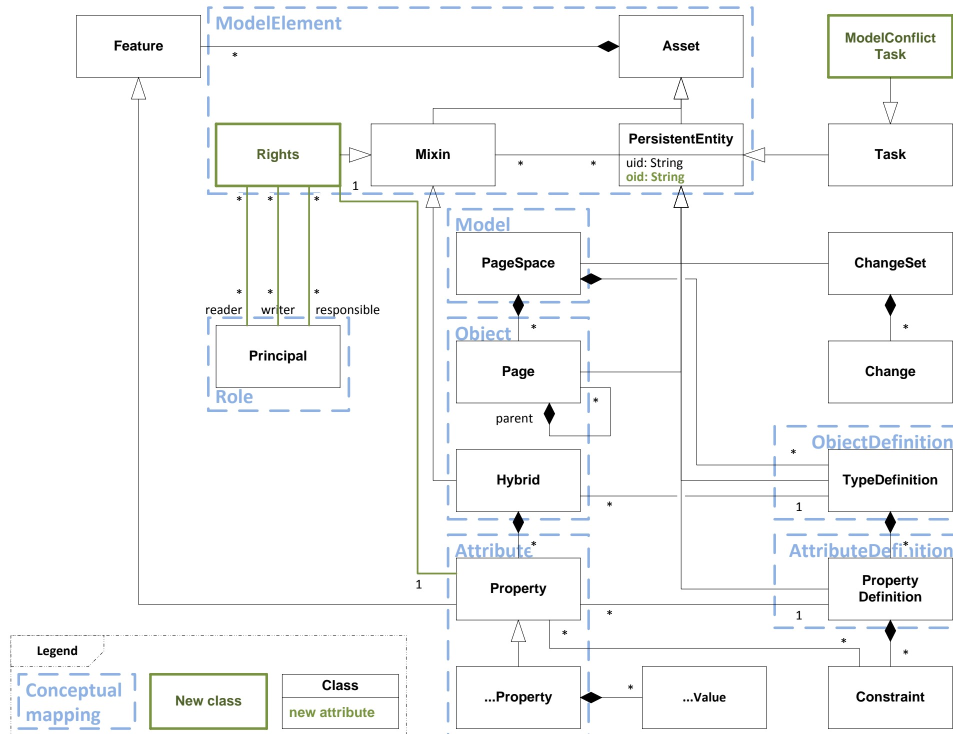


Figure 3.4.: Mapping of ModelGlue's meta-meta model onto Tricia concepts

in Tricia and do not inherit from *PersistentEntity*, they had to be directly linked to the *Rights* class.

This thesis will use conceptual terminology for explanations. Pictures of the ModelGlue implementation, however, may contain Tricia vocabulary.

The remainder of this section emphasises some further important features of the meta-meta model: the role concept and concepts of Model Conflict Tasks. Information about identifiers of Model Elements (oid and uid) can be found in section 3.4.2. Section 3.5.1 describes ChangeSets and Changes in detail.

3.3.1. Role concept

ModelGlue's meta-meta model (fig. 3.3) associates each model element with three different roles: read and write access as well as a responsibility role. This concept allows fine-grained access and responsibility control for models, objects, object definitions, attributes and attribute definitions. In contrast to our approach, most EA solutions only offer access rights on model and object level (see e.g. [Ne12]). Tricia, for instance, knows readers, writers (called 'editors'), administrators and contributors on model level and readers and writers on object level.

Why such detailed access rights? Given a federated organisation, some communities may or may not be allowed to read or alter information provided by other communities.

Example: The community engaged in information system (IS) modelling contributed an object 'SAP CRM application' to the EA model. As experts for this object, architects of the IS community have read and write access. The community for finance issues might add an attribute 'replacement cost' to the application object but restricting its visibility to only the head of the IS community. Write access could remain exclusively with the finance community.

Why the responsible role? Farwick et al. [Fa12] motivated the need for a responsible role for all model elements in the domain of EAM and particularly for automated EA model maintenance. The person or group responsible for an element of the EA model is the first contact person in case problems connected to this element should occur. He has extensive knowledge about the element and preferably also sufficient rights to solve potential conflicts.

Example: An object 'SAP CRM application' might have the attribute 'service level' and an 'application owner' as responsible person for this application. The attribute 'service level' specifies the role 'service level manager' as responsible and allows only this role to change values of the attribute. In case of a conflict related to the attribute 'service level', our fine-grained role model now allows to address not (or not only) the 'application owner', who is not allowed to change the 'service level', but rather the 'service level manager', who has more insights into this specific attribute and who is allowed to make changes.

Both examples laid out above can of course be generalised to schema concepts (object definitions and attribute definitions).

Default roles. Whereas users are familiar with access rights on object level, they may not always want to be bothered with specifying access rights to every attribute they create. Therefore ModelGlue follows clear role inheritance rules whenever a role has not been explicitly specified. In this case, object definitions inherit read and write rights (R/W) from the model. Objects and attribute definitions inherit access rights from the respective object definition. Attributes inherit access rights from their definition, if such a definition exists. Free attributes inherit access rights from the object they belong to. The user creating the element will become the default responsible role (RR). Figure 3.5 visualises this behaviour.

3.3.2. Model Conflict Tasks

A *Model Conflict Task* is a construct intended to describe a model conflict and provide the right users with the means to solve that very conflict, hence also called *conflict resolution task*. 'The right users' ideally refers to those users who have the deepest understanding of the conflict and are thus able to solve it. Section 3.3.2.4 details considerations about who to assign a task to.

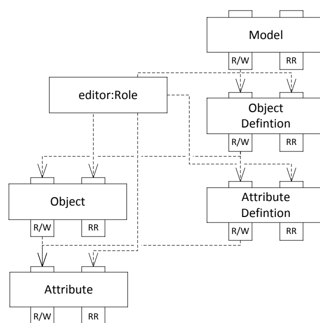


Figure 3.5.: Deduction of default roles

Figure 3.6 shows how *Model Conflict Tasks* are embedded into the static class environment of ModelGlue. ModelGlue is built upon an existing framework which already implements tasks for a manual assignment. Our new model conflict tasks inherit from these tasks. As our circumstances of EA model conflicts require more detailed task states (see section 3.3.2.2), the status attached to tasks was amended by a sub-status implemented in the class $ModelConflictTask$.

Every automatic model conflict task is triggered by a merge process. Consequently, this task knows the situation it was created in by maintaining a link to the corresponding $MergeData$ instance, which stores important meta information about every merge process (time, user, which models are involved).

The actual problem a model conflict task represents is implemented via the class $Conflict$ $InfoContainer$. A conflict info container knows all meta information about a model conflict as well as all elements and operations involved. Meta information includes a timestamp and the conflict classification (e.g. update object/delete object conflict on data level; see section 3.5.2.3 for details about conflict classification). Due to the one-to-one association, all this meta information could of course also be modelled directly in the model conflict task class. However, conflict info containers are also used as temporary data structure inside the merge algorithm. In this context the use of the bulky construct of tasks would add too much overhead, since most of a tasks concepts (assignees, ...) are not needed in early stages of conflict detection. Therefore we separated conflict meta information ($ConflictInfoContainer$) and the means for transportation and representation ($Task$).

Each conflicting version attached to a model conflict task is represented via a $ConflictList$ $Entry$. A conflict list entry is basically a tuple of a $Change$ and its corresponding $ChangeSet$. All information about the change operation is stored in the $Change$ class, information about the corresponding model element, change time, etc. can be retrieved from the $ChangeSet$ class (see figure 3.11). Since some occasions require that the conflict info container can easily distinguish which of the conflict list entries represents the base version, a direct link to this special conflict list entry (always exists exactly once) was implemented.

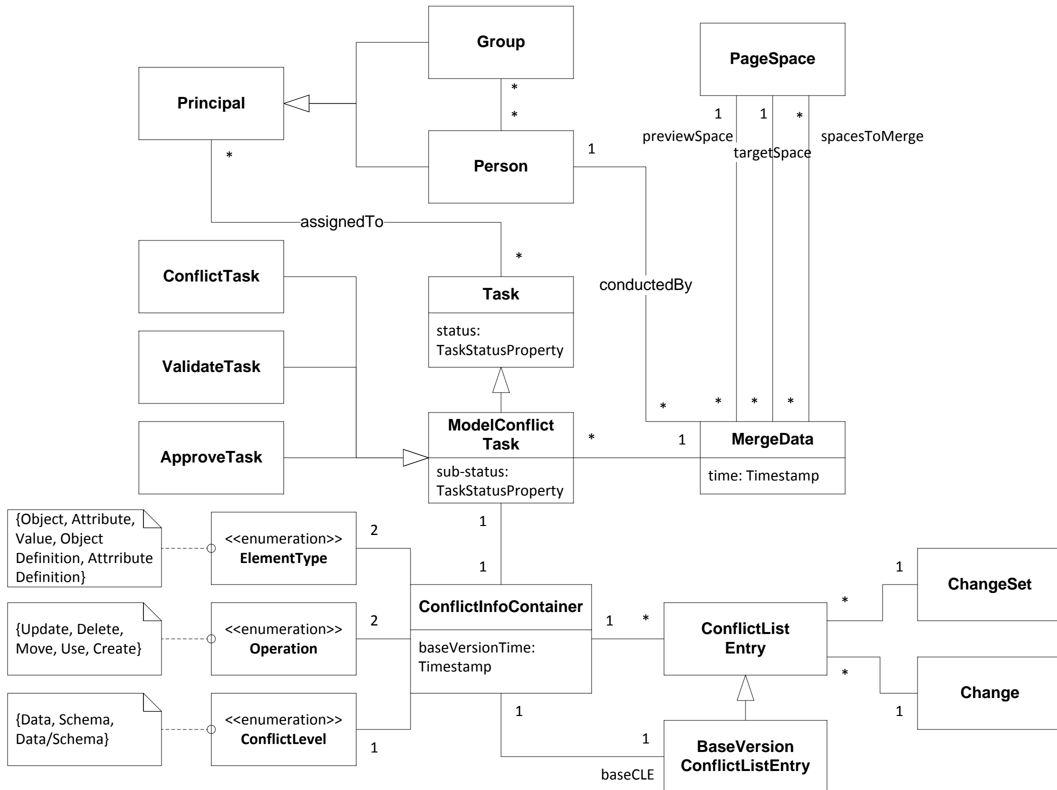


Figure 3.6.: Class diagram of the structure around Model Conflict Tasks

3.3.2.1. Task types

As published in [KR14], we distinguish between the following three model task types: *Conflicts*, *required approvals* and *required validations*.

Conflict tasks: Conflict tasks represent the typical scenario where changes in two or more elements cannot be resolved by an automated routine. One or possibly more responsible users have to decide which of the values is correct. Imagine an example where two users changed the same attribute of an object which is modelled in two information sources. When the changes do not coincide, the merge algorithm cannot decide which attribute to take. As it cannot resolve the situation without manual help, the algorithm triggers a conflict task in order to notify users involved in the conflict and to collaborative, manual solution.

Approve tasks: Deleting model elements could impact the EA model considerably. Therefore we introduced approve tasks in order enable responsible users to reconfirm delete actions which led to conflicts. Approve tasks always comprise a delete action which has to be approved or disapproved. Imagine a user having deleted an object 'Apache web server', while in another information source this very object was amended by a new attribute called 'next planned maintenance'. When merging the two sources, the algorithm cannot decide whether to keep the 'Apache web server' object (with the new attribute) or delete it and thus discard the attribute change. The approve task sent in this case urges users to solve this situation.

Since applying the new attribute requires to first decide about the deletion of the object, a delete operation is called the 'dominant' change in [KR14]. The operation conflicting with the 'dominant' change is called 'non-dominant'.

We anticipate three different resolution scenarios for approve tasks:
a) Users want approve the deletion and thus also discard the non-dominant change,
b) Users want to keep the non-dominant change and therefore also revert the deletion, or
c) Users change their mind concerning the deletion but still discard the non-dominant change.

Validate tasks: Validate tasks are used similarly as 'warning' tasks defined by Wieland et al. [Wi13]. They try to present situations with potentially unintended impacts on the semantics of an EA model. Changes conducted by one user might distort the semantics of other parts of the EA model.

By default, the merge routine applies all element versions involved in a validate task. The task is sent to allow users to revert one or several of these versions, if they really entail semantic problems. A typical scenario would be someone 'modelling a use-relationship from an application object to an object of type 'server', whose parent relationship classifies it as 'back-end server'. As another user moves this server from the 'back-end server' into the 'front-end server' domain in the other branch, it remains unclear whether the application object should still use the server under these semantically new circumstances' (cf. [KR14]).

These three tasks types are designed to being used for automated dissemination by a merge algorithm and to support conflict resolution. Other potential task types ModelGlue currently does not support but could be taken into consideration, particularly for manually triggered purposes, include 'assign role' (aim: delegate and refine roles) or 'document' (urges users to maintain data) tasks [RHM13a].

3.3.2.2. States of a Model Conflict Task

A model conflict can be either solved or not solved yet. But since a 'conflict' is an abstract construct, states can only be attached to the model conflict task which represents the actual conflict. Hence, on a high abstraction level, the task corresponding to the model conflict is either in state 'unsolved' or in state 'solved'. Figure 3.7 shows a state machine diagram of the states a model conflict task can have.

If the conflict is *solved*, it disappears from most user interfaces. In theory, the respective task could also be deleted. However, users may want to reconstruct actions performed on the EA model in the past, which includes conflict resolved via tasks. So we do not delete solved model conflict tasks but mark them as 'solved'.

The indicator 'unsolved' shows that a model conflict still exists, yet leaves many questions unanswered, including the following:

- Has anyone ever taken note of this conflict?

- Has the conflict task been delegated to someone who happens to be on holiday?
- How long has the conflict already been unanswered?
- Has the conflict been deliberately kept unsolved?
- Is the conflict irrelevant?
- Does something hinder users from solving this conflict?
- Is the conflict already being taken care of but the necessary changes require more time?

In order to clarify these questions, the state 'unsolved' needs to be categorised into sub-states. We distinguish between the sub-states 'new', 'reviewed', 'overdue' and 'ignored'.

Newly detected model conflicts are initially wrapped into tasks labelled as 'unsolved' with the sub-task 'new'. A *new* task has never been seen by anybody.
As soon as somebody opens the details view of a conflict (either in the conflict resolution dashboard or in the classical user interface), its state changes to 'reviewed'. This shows that someone has taken note of the conflict and is possibly working on a solution or has delegated the task.
Every model conflict tasks is sent with a 'due date' acting as a reminder and urging users to solve conflicts quickly. If the *due date* elapsed without the conflict being solved, the corresponding task turns into the sub-state 'overdue'. This allows sending reminder notifications as well as prominently emphasising due conflicts in visualisations and lists. If more time is needed for the conflict resolution, users can postpone the due date, thus transferring the task to the state 'reviewed' again.
Users may consider a conflict as irrelevant and therefore refrain from solving it. Then they would deliberately mark the conflict task as 'ignored'. Ignored model conflict tasks are presented less prominently in the user interface. Ignored tasks can of course be reactivated if the estimation of their significance changes. The concept of ignoring conflicts was a suggestion from company A's enterprise architect. Further reasoning on this topic can be found in section 4.1.6.

Since tasks in ModelGlue are persistent entities like for example pages in Tricia, they are subject to versioning control. All changes can be retraced in a change history log. Tasks can also be commented, which is important if several users are assigned to a task and involved in its resolution. Besides, commenting may prove useful if a user delegates the task to a colleague.

3.3.2.3. Impact of conflict tasks on the state of model elements

Model conflict tasks have impacts on those model elements they refer to. If a model element is afflicted with conflicts, it may be seen as a potential problem. Many conflict-afflicted elements in EA models are usually perceived as an indicator of low data/model quality.

In order to make this involvement in conflicts obvious, we distinguish the following model element states: 'clean', 'in conflict' and 'with ignored conflicts only'. A clean model element is not afflicted with any model conflicts or all corresponding conflict tasks have already been

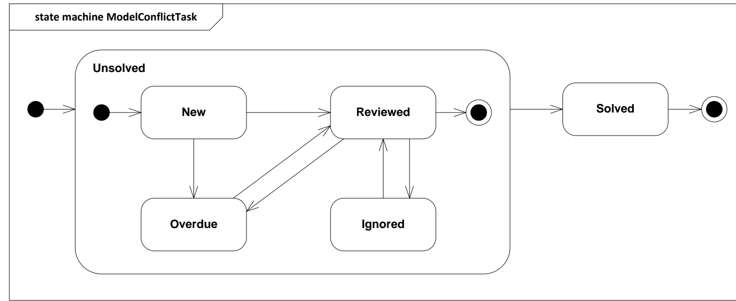


Figure 3.7.: States of a Model Conflict Task

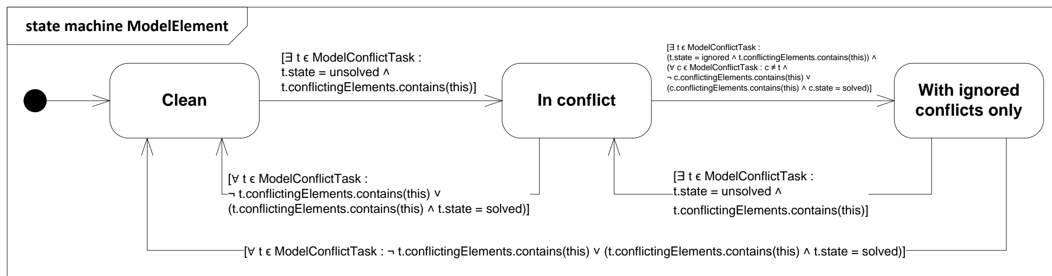


Figure 3.8.: States of a Model Element

solved. As soon as a model conflict task is associated with a model element, the state of this element changes to 'in conflict'. This allows to highlight model elements afflicted with conflicts. If all conflicts related to a model element are 'unsolved', but with the sub-state 'ignored', this model element is classified via the state 'with ignored conflicts only'. The differentiation between 'in conflict' and 'with ignored conflicts only' allows precise impact estimations of ignored conflicts. Elements 'with ignored conflicts only' may not be an immediate problem for the EA model but they still contain unsolved issues and are hence not quite as clean as 'clean' elements are.

3.3.2.4. Task assignment

As stated before, Model Conflict Tasks are meant to reach those persons or groups who have a deep understanding of the conflict (*requirement a*) as well as the permission to solve the conflict (*requirement b*). Task assignees should be unbiased and hence incorporate positions with a sightly different viewpoint than the last editor who implicitly triggered the conflict (*requirement c*). Not least because of a potentially high amount of conflicts triggered by a single merge process should the number of task recipients remain within reasonable bounds (*requirement d*).

Apart from read (R), write (W) and responsible (RR) roles modelled in our meta-meta model, the last editor (E) of an element can be taken into consideration. These roles are mutually related and subject to the following mathematical rules: $RR \subseteq W$; $E \subseteq W$; $W \subseteq R$; $|RR| \ll |W|$; $|W| \ll |R|$; $|E| = 1$;

This section describes a few possible task assignment scenarios:

- By **default**, a task embracing model elements e_1 and e_2 is sent to the responsible roles of both elements ($e_1.RR \cup e_2.RR$). Due to $RR \subseteq W$, this strategy ensures that requirement *b*) is fulfilled. Assuming RR $\neq$ E, requirement *c*) is fulfilled, too. As it makes little sense to assign the responsible role to a large group of users, requirement *d*) should not pose a problem. Requirement *a*) cannot be guaranteed, since a responsible person might well be in a position with little contact to the abstraction level adequate for conflict resolution.
- A task could be assigned to the last editors of the respective elements ($e_1.E \cup e_2.E$). This strategy fulfils requirement *a*), *b*) and *d*), but violates requirement *c*).
- A task could be assigned to all writers of the respective elements ($e_1.W \cup e_2.W$). Similar to the previous strategy, this approach fulfils requirement *a*) and *b*). As many persons are addressed, it also fulfils requirement *c*). The main problem here is that a lot of persons are bothered (requirement *d*)). This case considers the task as solved as soon as one assignee solves the respective conflict.
- Assignment, like in the previous item, to $e_1.W \cup e_2.W$. The task, however, will only be marked as solved when a certain quota (e.g. 51%) of assignees has approved of the conflict solution provided by the first person who solved the conflict. Especially if the conflict consists of approving the deletion of an element (see section 3.5.2.3), such voting solutions may prove promising.

This section motivates to explore the diversity of collaborative potential. All reasoning expressed within this section - both the justification of requirements and the appropriateness of assign solutions - is yet to be appraised in further, more technical, evaluation steps.

3.4. Model branching

As stated by Achenbach [Ac13], EAM relies on modelling planned and target states of the enterprise architecture model.

To derive such planned or target states, we adapted concepts and terminology from code versioning systems like Subversion or GIT. Technically, such *model branching* is conducted by simply creating a deep copy of the initial model.

One challenge, however, is how to store and guard information that the branched model is a copy of all objects within the original model. Such ties are vital in a realistic environment where EA models co-evolve. This section distinguishes between meta-information about the branch process stored only on model level and ties kept in all elements within the model.

3.4.1. Branch meta-information

Whenever a model is branched, three additional pieces of information are stored as meta-information in the new branch: The Id of the source model, the user who triggered the

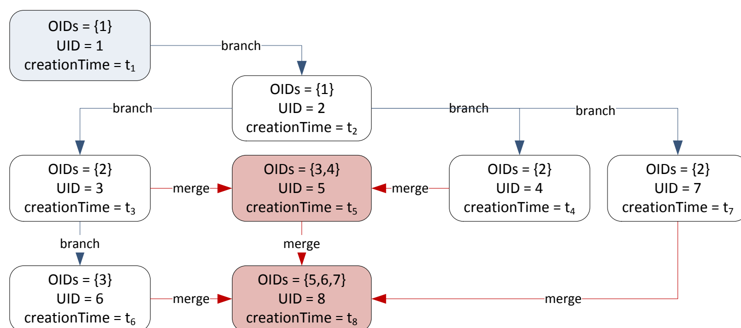


Figure 3.9.: Original identifier (Oid) evolution of models

branch and a timestamp storing the branch creation time, i.e. the time the branch process was finished.

A new model branch may also be the result (= target model) of a merge process (see section 3.5.2). In contrast to manually branched models, merge target models have several source models - at lease two and in general n, since we pursue an n-way merge approach.

Therefore information about the source model(s) of a branched or merged model has to be stored in a list. This list comprises the Ids of all direct predecessors of the new branch. Values in this lists are referred to as **original identifiers** or simply **Oids** (cf. section 3.4.2).

Meta-information about branches allows to keep an overview of all models created in an enterprise, especially of the question: which model derived from which other models. With this information, even a visualisation illustrating the model evolution graph, similar to figure 3.9 could be integrated into the user interface.

Most important, however, is the timestamp. In the context of co-evolution, it allows to discern the last common state of two related models. This last state is referred to as *base version*. Accordingly, the timestamp of the base version is called *baseline time* or simply *base time* t_B.

3.4.2. Ties between model elements

Apart from meta-information on model level described in the previous section, we want to keep track of all ties between model elements across branches. Assuming model B is a branch of model A, we want to maintain knowledge of which element e_B has been branched from which element e_A and thus at time t_B had been an exact copy of e_A.

This knowledge is achieved via a concept called **original identifier** or simply **Oid**. As element Uids (unique identifiers) have to remain unique across all models, they cannot be copied along with everything else. So for each copied model element the branching algorithm of course assigns a new Uid. However, it sets the property 'Oid' of the branched element to the 'Uid' of the original one. This way every branched element stores a link to its archetype. For each model element created after t_B in one of the branches, the Oid will be put to *null* because this new element has no counterpart in any other model.

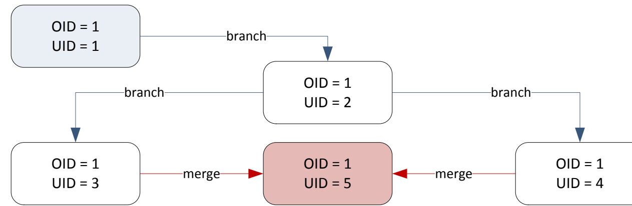


Figure 3.10.: Original identifier (Oid) evolution of model elements

Why do we need Oids? Storing ties between elements across related models is a prerequisite for operation-based merging (as introduced in section 3.5.2). If for instance a user wants to unify two related EA states, the merge algorithm is able to distinguish between elements sharing the same origin and newly created elements. To a certain extent this would also be possible via name comparisons (without Oids), but only with some severe disadvantages delineated in section 3.5.1.

Oids are stored for all model elements of ModelGlues meta-meta model (figure 3.3), i.e. models, objects, object definitions, attributes and attribute definitions. Depending on the required depth of operation detection, Oids could also be stored on value level. This would facilitate operation detection on value level (e.g. how to discern 'create value' from 'update value') and could fill some of the empty cells in the operation detection table 3.1. However, we deemed Oids on value level not essential in the context of EA models. Thence Oids on value level are not implemented in ModelGlue.

Data structure for Oids on model level: Main purpose of Oids on model level is the construction of a model evolution graph the calculation of the baseline time needed in merge and differences algorithms. Just storing one Oid for every model would allow evolution trees. However, merge operations generally cause evolution graphs. Hence, Oids of a model have to be stored as a list, like shown in figure 3.9.
The Oid list of a new model comprises the Uids of all its immediate source models.

Data structure for Oids on model element (not model) level: Main purpose of Oids on model element level is the detection of cross-model ties between elements, which is needed in merge and differences algorithms. Saving a list of Oids in every model element would allow the construction of an evolution graph for every model element and of course fulfil every desired functionality. But traversing the entire evolution graph only to detect whether two elements e_1 and e_2 share common ancestors is inefficient. Therefore ModelGlue stores exactly one Oid and not a list of Oids in model elements (only a list of Oids in the model itself). Figure 3.10 illustrates this data structure.
If a model element is the result of a branch process, the Oid of its predecessor is saved as its own Oid. In case of a merge process, inheriting the Oid from any one of the source models is sufficient. As can be seen in figure 3.10, e.g. the elements with Uids '4' and '5' share the same Oid and are hence related.
Taking only the Oid into account, it is impossible to discover that not element '1' but element '2' is the last shared state of elements '4' and '5'. However, since the whole evolution

graph can be reconstructed via the meta-information attached to the corresponding model elements, such detailed information does not need to be stored on every level. Saving the information once on model level is sufficient for all use cases presented in this thesis.

3.4.3. The baseline time

For merge and differences algorithms presented in this work (sections 3.5.2 and 3.6) it is vital to know whether two models are related and what the last common state of these models is. As Tricia, the underlying system ModelGlue was built on, allows the reconstruction of model states at any time in the past, the only information necessary for this reconstruction is the exact creation time of the last common state (also called 'base version'). The creation time of the base version is called 'baseline time' or 'base time' or 't_b' throughout this thesis.

Calculation of the baseline time t_b for n input models $M_1, ..., M_n$ can be achieved via the following algorithm:

Algorithm 1: Calculates the baseline time t_b of the models $M_1, ..., M_n$

Data: Input models $M_1, ..., M_n$
Result: Baseline time t_b

```
1  predecessors_M1, ..., predecessors_Mn ← {∅}
2  foreach M_i ∈ M_1, ..., M_n do
3      predecessorTree_Mi ← generatePredecessorTree(M_i)
4      // generatePredecessorTree(M_i) generates the model evolution graph of M_i. However, it
           considers only incoming edges and takes M_i as root node. Therefore the resulting graph is
           a tree.
5      predecessors_Mi ← calculateListOfPredecessors(predecessorTree_Mi)
6      // calculateListOfPredecessors(predecessorTree_Mi) conducts a breadth-first search in the
           tree generated above. It returns a list of all child nodes of this tree.
7  foreach predecessors_Mi ∈ predecessors_M1, ..., predecessors_Mn do
8      predecessors_Mi ← predecessors_Mi.sort()
9      // Sorts the list of all predecessors M_p of model M_i so that those M_p with the
           latest/newest creation time come first.
10 foreach M_p ∈ predecessors_M1 do
11     if M_p ∈ predecessors_M2 ∧ ... ∧ M_p ∈ predecessors_Mp then
12         return t_b ← M_p.creationTime
13         // Since all predecessor lists are sorted according to model creation time, the first
               solution is the latest/newest and thus most precise solution.
14 return ∅
15 // If the algorithm reaches this point, no common base version exists.
```

The following paragraph will explain algorithm 1 in case of the scenario illustrated in figure 3.9.

Imagine we want calculate the baseline time of the model with Uid '8', which is the result of a merge between models '6', '5' and '7'. They will in the following be referred to as M_6, M_5 and M_7.

At first, predecessor lists for these three models are calculated. $predecessorTree_{M_6}$ is a tree with M_6 as root node and M_3, M_2 and M_1 as child nodes (with edges $6 \rightarrow 3$, $3 \rightarrow 2$ and $2 \rightarrow 1$). Consequently, $predecessors_{M_6}$ is the list $\{M_3, M_2, M_1\}$. Predecessors for M_5 and M_7 are calculated accordingly ($predecessors_{M_5} = \{M_3, M_4, M_2, M_1\}$; $predecessors_{M_7} = \{M_2, M_1\}$).

The next step sorts all predecessor lists such that newer predecessors appear prior to older predecessors. The lists for M_6 and M_7 already fulfil this order; $predecessors_{M_5}$ is sorted to $\{M_4, M_3, M_2, M_1\}$.

Finally, iterating through all items in $predecessors_{M_6}$, a common base version is searched for. The first item (M_3) is not included in $predecessors_{M_7}$, so it cannot be the base version. The second item (M_2) appears in all lists, so M_2 must be the base version of the three models M_6, M_5 and M_7 given as input.

Baseline time t_b is the creation time of model M_2.

3.5. Model merging

Model merging is the heart of all ModelGlue functionality, hence also the expression Model-'Glue'. In order to fulfil all use cases related to merging (introduced in section 3.2 at the beginning of this chapter), several important prerequisites have to be met. This section starts with a report on fundamental design decisions (section 3.5.1), continues with a detailed analysis of our merge algorithm (section 3.5.2) and ends with conflict classification, resolution and learning capabilities (sections 3.5.3 to 3.5.5).

3.5.1. Operation-based approach

The first fundamental task before actually implementing a merge routine is to determine and specify how different models can be compared. This depends on how model changes are conceptually represented, physically stored and how they can be detected afterwards (e.g. for a merge).

Literature generally distinguishes between state-based and operation-based (a subset of change-based approaches [CW98]) model comparisons [Ko10].

State-based approaches take two models as an input and compare them by trying to detect differences between these static models. For this differences detection, usually the whole static structure of the model has to be parsed [CW98]. Especially for large models, investigating the whole structure if often both time-consuming and unnecessary.

When applied to typical operations within EA models, we anticipated some situations that cannot be detected by a state-based approach. Among others, this approach has severe disadvantages in the following cases (Assumption: e_B is the branch of e_A):

- Renaming e_A or e_B leads to a loss of the link to the counterpart.
- Delete operations cannot be detected.
- Move operations cannot be detected.
- The algorithm cannot discern the deletion of e_A from the creation of e_B.
- Adding new elements in both branches which coincidentally have the same name will be treated as though they are related.

In contrast to state-based approaches, **operation-based merging** does not take the input models themselves but their change operations as input. These operations are then combined into a single output model. During this process, conflicts can be detected and potentially also resolved [LVO92].

Due to the mentioned problems of state-based merging and the well-documented advantages of operation-based solutions, particularly in the field of conflict detection and resolution

potentials [LVO92, KHS09], ModelGlue's merge algorithm applies an operation-based approach. In doing so, we follow a general trend towards operation-based merging identified by Mens [Me02].

Operations in ModelGlue: Similar to Wieland et al. [Wi13], who investigated model changes in the field of software engineering, we distinguish between five different types of changes executed on model elements: **create, delete, update, use** and **move**. The first three operations are commonly well-known. *Use* describes the situation when one model element e_1 establishes a link to another element e_2. e_1 then *uses* the resource e_2. A *move* exists when a model element changes its context. For *Objects*, this might be its position in the *Object* hierarchy as well as its belonging to a certain *ObjectDefinition*.

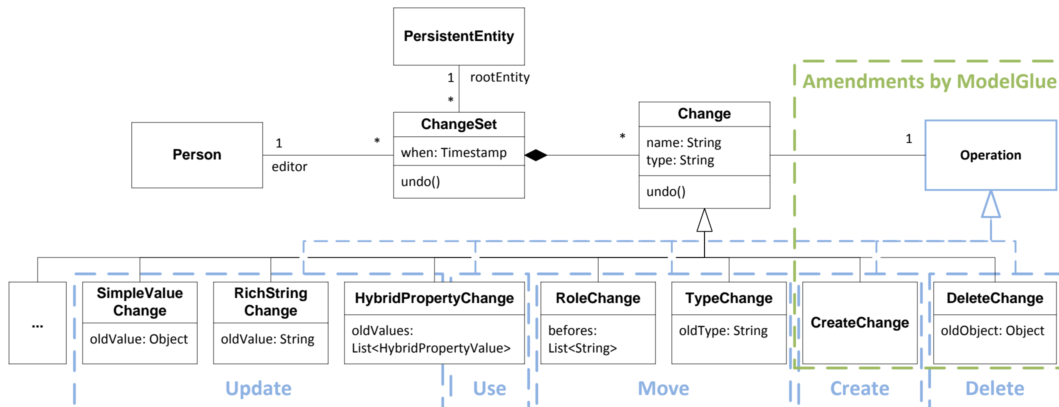


Figure 3.11.: Class structure around *ChangeSets* and mapping of *Change* classes to conceptual ModelGlue operations (blue boxes)

Change data structure in Tricia: Since Tricia does not explicitly implement the exact five change operations mentioned above, it had to be adapted in order to support ModelGlue's operation-based merge approach. Figure 3.11 depicts Tricia's class structure around the two constructs representing model changes, *ChangeSet* and *Change*.

ChangeSet: A ChangeSet stores all meta-information relevant when changing model elements. It saves time and date of the action, a link to the model element that was changed (*rootEntity*) as well as a link to the person who performed the change. Note that on implementation level, the Tricia term *PersistentEntity* is a superclass of ModelGlue's *ModelElement*. Similar to *transactions* in the domain of database technology, a *ChangeSet* contains a bundle of several *Changes*. Of the typical transaction characteristics atomicity, consistency, isolation and durability (ACID; see e.g. Kemper and Eickler [KE09], p. 283), Tricia's *ChangeSet* concept heeds consistency and durability in particular. Changing e.g. several values at once in one user mask would be deemed one *ChangeSet*. Following the principle of consistency, only whole *ChangeSets* may be reverted ('undo' functionality).

Change: Actual model alterations are stored in the Change class. 'Name' shall answer the question of what aspect of the *ChangeSet*'s *rootEntity* was changed. If the *rootEntity* is an *Object*, 'name' might e.g. contain the name of the object's attribute that was changed. Given to the multitude of different information formats that can be stored in Tricia, 'type' allows fine-grained change format identification. 'Type' is actually a String representation of the respective *Change* subclass name, e.g. 'HybridPropertyChange'. The *Change* class also stores all information necessary for undoing itself as well as the functionality for this reversion. Consequently, it always saves the old value before changing an element to a new value. Due to the different value formats, this property is called 'oldValue' in those *Change* subclasses which deal with single values and slightly different in most other subclasses.
Figure 3.11 shows the subset of *Change* subclasses that is relevant for EA models.

Example: Figure 3.12 illustrates an example where the user 'Seppl' changed the name of an element 'JBoss' from 'jboss7_x64' to 'JBoss 7.1.1'. *ChangeSet cs* stores him as editor and a reference to the element 'JBoss', as well as date and time of the action. Changing the single, primitive value of an element is considered a $SimpleValueChange$ in Tricia, hence this change *type*. $Name$ of the change happens to be 'name', as that is the name of the changed property. The *Change* class of course also saves the old value for potentially reverting the operation. *Operation* is a complement introduced by ModelGlue - it discerns this $SimpleValueChange$ as an *update* operation in ModelGlue's conceptual terminology.

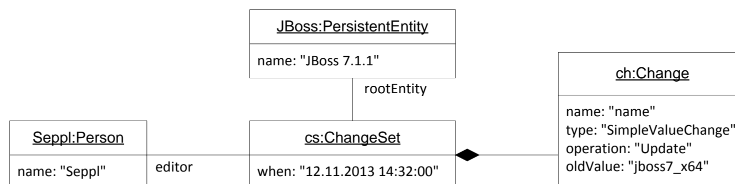


Figure 3.12.: ChangeSet and Change in an update example

Amendments by ModelGlue: Providing a *Change* with an *Operation* is an amendment implemented in order to map the many *Change* subclasses onto the five conceptual ModelGlue operations (create, delete, update, use and move). As the blue boxes in figure 3.11 illustrate, in some cases the conceptual mapping exclusively depends on the *Change* type. Some changes, however, require further information. Tricia's standard implementation models *create* and *delete* operations without using the concept of *Change* classes. So in order to attain an exhaustive and homogeneous basis for model merge functionality, *CreateChange* and *DeleteChange* had to be appended in the class hierarchy.

Mapping of change types to ModelGlue's operations: The following list details the technical mapping necessary to identify the five conceptual operations *create, delete, update, use and move.* Table 3.1 summarises the mapping approach. Note that some cases can not be modelled in Tricia.

- **Create Object**: Detecting a *create object* operation is rather straightforward. The creation initiates a new *ChangeSet* which contains only its own meta-information

and no *Changes*. Since ModelGlue's merge algorithm always requires a tuple of *ChangeSet* and *Change* as input for conflict detection, we amended Tricia by a new class *CreateChange*. **Create ObjectDefinition** and **create AttributeDefinition** work likewise.

- **Delete Object**: Whenever a user deletes an object, it is removed from the database table where all existing objects are stored and shifted to a database table dedicated for deleted entities (similar to the recycle bin in most operating systems).
  Tricia first creates a new instance of class *Deleted*, which stores meta-information of the delete operation (user and time). It also wraps a serialised version of the deleted *PersistentEntity*, which can be restored at any time. As it stores all information required in a delete scenario, Tricia's *Deleted* class comes close to being able to serve as an implementation of ModelGlue's delete operation. A big problem, though, is that at first glance *Deleted* instances reside in their dedicated database table, isolated from the currently existing model. The meta-information of *Deleted* elements does not suffice to discern ties of the deleted object to other model elements which still exist. Owing to this lack of context information, the *PersistentEntity* wrapped within **the *Deleted* entity has to be temporarily restored** in order to be examined by the conflict detection algorithm. Consequently, the whole *Deleted* database has to be queried when trying to detect conflicts related to deleted model elements.
  Note that these restored element serve only for algorithm-internal purposes and are not shown on the user interface. They will be erased as soon as they are not needed any longer. That is the case when the merge routine finishes without a particular deleted object being involved in any conflicts, or when a task involving it is solved. For the sake of homogeneity required in the merge algorithm, ModelGlue creates a *ChangeSet* and a *DeleteChange* instance for every temporarily restored element.
  Delete ObjectDefinition and **delete AttributeDefinition** are detected alike since object definitions and attribute definitions are also subclasses of *PersistentEntity* (see figure 3.4).

- **Update Object**: In Tricia, three cases can be identified as an *object update*: Every *PersistentEntity* in Tricia may have *Properties*[5] (see figure 3.4). One object property is for example 'show tasks', which can be enabled or disabled. Changing such a property is stored as a *SimpleValueChange* (**a**) for properties with singular values and a *RichStringChange* (**b**) for formatted strings. The third possible *object update* is to change one of its roles, i.e. either the responsible role or writers or readers. Such a change is saved as *RoleChange* (**c**).
  Update ObjectDefinition, **update AttributeDefinition** and **update Attribute** function correspondingly.

- **Use Object**: Since relationship information about an *Object* is stored via *Relationships* (also called *LinkValues*; see section 3.3), detecting *use* operations requires profound examination. It occurs when a new relationship value linking to another *Object* is added to an *Attribute* of the *Object* under observation.

[5] In-depth analysis of Tricia's data modelling framework can be found in Neubert's ([Ne12], p. 86ff) and Büchner's ([Bü07], p. 138) work.

Table 3.1.: Mapping of Tricia changes to ModelGlue operations

	create	delete	update	use	move
Object	ChangeSet without Change	Querying the 'Deleted' database	SimpleValueChange, RichStringChange, RoleChange (for changing rights)	HybridProperty Change for LinkValues to other *Objects*	TypeChange, RoleChange (for the role 'parent')
Object Definition	ChangeSet without Change	Querying the 'Deleted' database	SimpleValueChange, RichStringChange, RoleChange (for changing rights)	SimpleValueChange of an AttributeDefinition, if the new value is a constraint to another ObjectDefinition	-
Attribute Definition	ChangeSet without Change	Querying the 'Deleted' database	SimpleValueChange, RichStringChange, RoleChange (for changing rights)	-	-
Attribute	HybridProperty Change c, if $c.oldValues \equiv \varnothing$	HybridProperty Change c, if c.changeSet.root Entity.hasProperty (c.name)	SimpleValueChange, RichStringChange, RoleChange (for changing rights)	-	-
Value	HybridProperty Change c, if $c.oldValues.size \leq$ c.changeSet.root Entity.getProperty (c.name).values.size	HybridProperty Change c, if $c.oldValues.size \geq$ c.changeSet.root Entity.getProperty (c.name).values.size	HybridProperty Change, if not identified as a *create* or *delete* operation	-	-

Despite the fact that the actual change happens to an *Attribute*, it is still regarded as an *useobject* operation because the destination of the relationship can only point to another *Object*.

- **Use ObjectDefinition**: This case works similarly to the *use Object* case, with the difference that instead of an *Attribute* an *AttributeDefinition* has to contain a new constraint to another *ObjectDefinition* (SimpleValueChange).
- **Use AttributeDefinition, use Attribute, use Value**: Since Tricia only permits links to *Objects* or *ObjectDefinitions*, the ModelGlue implementation does not cover these cases. In general, for *AttributeDefinitions* and *Attributes*, *use* operations might well be taken into consideration. *Use Value* is probably too fine-grained to be of substantial value.
- **Move Object**: The operation *move* intends to describe context changes of an *Object*. Such a context change may occur along two dimensions: Firstly, positioning an *Object* under a different *ObjectDefinition* (*TypeChange*) is regarded as a *move*. And secondly, relocating an *Object* in the *Object* tree of a model (*RoleChange*, where 'role' = 'parent') is classified likewise.
- **Move ObjectDefinition**: Due to the flat Tricia *ObjectDefinition* structure (no hierarchy), they cannot be moved. Despite this limitation of Tricia - in general, moving object definitions should be taken into consideration.
- **Move AttributeDefinition**: If Tricia allowed moving an *AttributeDefinition* from one *ObjectDefinition* to another, this case might be relevant. But the necessary functionality is currently not implemented.
- **Move Attribute**: Like their definitions, *Attributes* cannot be moved in Tricia.
- **Move Value**: *Values* cannot be moved to other *Attributes* in Tricia. Hence, this case is not pursued any further. Conceptually, this operation might be considered, but it

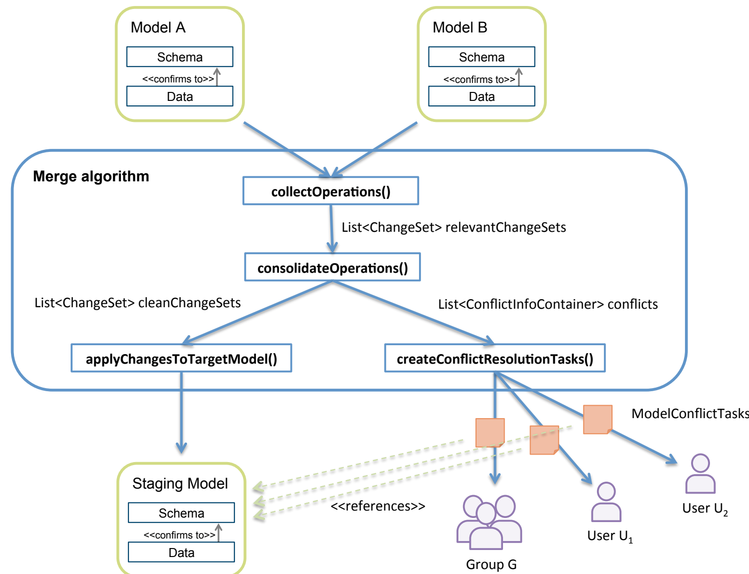


Figure 3.13.: Overview of the merge algorithm

remains to be evaluated if such a fine-grained case would lead to an excessive amount of detected operations.

- **Create Attribute**: A *HybridPropertyChange* where 'oldValues' is *null* (see class diagram 3.11) indicates a *create Attribute* scenario.
- **Delete Attribute**: A *HybridPropertyChange c* stores the name of the changed *Attribute* in *c.name*. If the corresponding *Object* does not contain any *Attribute* called *c.name*, this *Attribute* must have been deleted.
- **Create Value, delete Value**: If after a *HybridPropertyChange* the number of *Values* of the respective *Attribute* is higher/lower than before, we assume that at least one value has been created/deleted. ModelGlue deliberately does not implement more precise handling capabilities for *Values* as they were not deemed necessary. If more control on this fine-grained abstraction level is needed, the *Value* class would have to become a subclass of *ModelElement* and thereby profit from unique identifiers.
- **Update Value**: Every *HybridPropertyChange* that is not classified as any other case (see table 3.1) is labelled as an *update Value* scenario. Like all *Value* operations, this classification is rather imprecise and could be improved by providing *Values* with IDs.

3.5.2. Model merging

Figure 3.13 gives a high-level overview of the approach pursued for model merging. Given two (like depicted) or more EA models as input, an algorithm merges all non-conflicting changes into one preliminary target model (also called staging model) and generates Model Conflict Tasks for all conflicting changes.

Internally, this merge algorithm is divided into the key functions *collectOperations()*, *consolidateOperations()*, *applyChangesToTargetModel()* and *createConflictResolutionTasks()*.

Heart of the merge algorithm is its conflict detection and the classification routines. This part is implemented in the function *consolidateOperations()*. Conflict detection and classification is structured into three steps:

1. **Element pre-selection** $\rightarrow$ Only consider connected elements, which could potentially be conflicting. See section 3.5.2.1.
2. **Conflict detection** $\rightarrow$ Is this case really a conflict? See section 3.5.2.2.
3. **Conflict classification** $\rightarrow$ Classify the case into one of the task types (conflict, approve or validate). See section 3.5.2.3.

In the following sub-sections, these three steps will be explained in detail. Subsequently, the whole merge algorithm is examined (section 3.5.2.4).

3.5.2.1. Step 1: Element pre-selection

Conflict detection works via pairwise comparisons of changesets and included changes. But investigating all pairs of changesets cs_1 and cs_2 within the Cartesian product of all changesets of the respective models would be a very time-consuming effort due to its exponential complexity ($\mathcal{O}(n^2)$). Imagine two models with 1500 model elements, each having been changed three times; this scenario would result in $(1500 * 3)^2 \approx 20$ million comparisons. Luckily, investigating all these possibilities is also an unnecessary effort, since most comparisons will be between changes of elements which are in no way connected or related to each other.

Consequently, a pre-selection has to be conducted before proceeding with *step 2: detection and classification*. This pre-selection examines connections between the root entities[6] e_1 and e_2 of changesets cs_1 and cs_2. In the code, the method conducting this pre-selection is therefore called **areConnected(**ModelElement e_1, ModelElement e_2**)**. Element pairs being detected by this method are denoted as $\boldsymbol{e_1 \bowtie e_2}$ throughout this thesis.

Pre-selection operates similarly to experts systems [Ja86]. It eliminates all tuples (cs_1, cs_2) which can not be allotted to one of the following cases:

1. $\boldsymbol{e_1 \cong e_2}$: The congruence symbol $\cong$ is used to indicate that one element is a branch of the other or both have been branched from the same source.If $e_1 \cong e_2 \Leftrightarrow true$, we call e_1 and e_2 **'related'**. In the code, such relations are detected via the method **areRelated(**ModelElement e_1, ModelElement e_2**)**.

$$\begin{aligned} \boldsymbol{e_1 \cong e_2} \Leftrightarrow & (e_1.uid \equiv e_2.oid) \\ & \vee (e_1.oid \equiv e_2.uid) \\ & \vee (e_1.oid \equiv e_2.oid) \end{aligned} \tag{3.1}$$

[6] see figure 3.11 for the term 'rootEntity'

2. $\boldsymbol{(e_1.name \equiv e_2.name) \wedge (e_1.\text{`}context' \cong e_2.\text{`}context')}$: Name collisions. Elements with identical names lead to conflicts whenever they are modelled within the same context. 'Same context' refers to the same hierarchy level (identified via the 'parent' link in class diagram 3.3) for *Objects*, the same *ObjectDefinition* for *Attribute Definitions*, and the same *Object* for *Attributes*. Due to ModelGlue's flat type structure, *ObjectDefinition* names must globally be unique. For the sake of brevity, in the following equations *ObjectDefinition* will be abbreviated as *ObjDef*, *Attribute* as *Attr*, and so forth.

$$\begin{aligned}
&\boldsymbol{(e_1.name \equiv e_2.name) \wedge (e_1.\text{`}context' \cong e_2.\text{`}context')} \\
&\Leftrightarrow (e_1.name \equiv e_2.name) \\
&\wedge \Big((e_1 \; instanceof \; ObjDef \wedge \; e_2 \; instanceof \; ObjDef) \\
&\vee \big((e_1 \; instanceof \; AttrDef \wedge e_2 \; instanceof \; AttrDef) \\
&\wedge (e_1.objectDefinition \cong e_2.objectDefinition)\big) \\
&\vee \big((e_1 \; instanceof \; Obj \wedge e_2 \; instanceof \; Obj) \wedge \; (e_1.parent \cong e_2.parent)\big) \\
&\vee \big((e_1 \; instanceof \; Attr \wedge e_2 \; instanceof \; Attr) \wedge \; (e_1.object \cong e_2.object)\big)\Big)
\end{aligned} \tag{3.2}$$

3. $\boldsymbol{e_1.type \cong e_2.type}$: Both elements are *Objects* or *AttributeDefinitions* and confirm to *ObjectDefinitions* which are related in terms of rule 3.1 or 3.2.

$$\begin{aligned}
\boldsymbol{e_1.type \cong e_2.type} \Leftrightarrow &\big((e_1 \; instanceof \; Obj \wedge e_2 \; instanceof \; Obj) \\
&\wedge (e_1.type \cong e_2.type)\big) \\
\vee &\big((e_1 \; instanceof \; AttrDef \wedge e_2 \; instanceof \; AttrDef) \\
&\wedge (e_1.objectDefinition \cong e_2.objectDefinition)\big)
\end{aligned} \tag{3.3}$$

4. $\boldsymbol{e_1 \cong e_2.type}$: Assumption: e_1 is an *ObjectDefinition* and e_2 an *Object* or an *AttributeDefinition*

$$\begin{aligned}
\boldsymbol{e_1 \cong e_2.type} \Leftrightarrow &\; e_1 \; instanceof \; ObjDef \\
&\wedge \big((e_2 \; instanceof \; Obj \wedge e_1 \cong e_2.type) \\
&\vee (e_2 \; instanceof \; AttrDef \wedge e_1 \cong e_2.objectDefinition)\big)
\end{aligned} \tag{3.4}$$

In summary, we anticipate potential conflicts if the following equation (3.5) is true. If $e_1 \bowtie e_2 \Leftrightarrow true$, we call e_1 and e_2 **'connected'**.

$$\begin{aligned} &\boldsymbol{e_1 \bowtie e_2} \\ &\Leftrightarrow e_1 \cong e_2 \\ &\vee (e_1.name \equiv e_2.name) \wedge (e_1.\text{`}context' \cong e_2.\text{`}context') \\ &\vee e_1.type \cong e_2.type \\ &\vee e_1 \cong e_2.type \end{aligned} \tag{3.5}$$

3.5.2.2. Step 2: Conflict detection

After all insignificant changesets have been sorted out by the element pre-selection (step 1), this step identifies whether two given changes really are conflicting. Whereas step 1 only considered whether two model elements are connected and thus could in theory cause a conflict, step 2 now also considers reciprocal effects between changes and their respective elements.

Line 12 in algorithm 2 detects a conflict if the following expression is *true*:

$$(cs_1.rootEntity \not\equiv_{ch_1} cs_2.rootEntity) \vee (cs_1.rootEntity \not\equiv_{ch_2} cs_2.rootEntity) \tag{3.6}$$

The equivalence symbol $\equiv_{ch_x}$ with a *Change* index ch_x reads as follows: '...are equal with regard to change ch_x'.

Imagine change ch_x representing a change of the object property 'name'. Now if two objects o_1 and o_2 have different names, they are deemed not equal with regard to ch_x: $o_1 \not\equiv_{ch_x} o_2$

If o_1 and o_2 have identical names but differ in the attribute 'attrXYZ', they are certainly not equal ($o_1 \not\equiv o_2$). However, they are deemed equal with regard to ch_x: $o_1 \equiv_{ch_x} o_2$

ModelGlue's conflict detection always compares two tuples of changeset and change, i.e. compares (cs_1, ch_1) with (cs_2, ch_2). The change stores information about what happened, i.e. the change operation. The changeset knows all necessary meta-information of the change, for instance the link to the changed model element $e_1 = cs_1.rootEntity$. As ch_1 and ch_2 might possibly be different operations and yet cause a conflict (e.g. update/delete), equivalence with regard to both ch_1 and ch_2 has to be tested.

If equation 3.6 (resembling line 12 in algorithm 2) is *true*, the pair of tuples $\big((cs_1, ch_1), (cs_2, ch_2)\big)$ is marked as a conflict. In a subsequent step, this case now has to be classified.

3.5.2.3. Step 3: Conflict classification

Conflict classification within ModelGlue is inspired by Wieland et al., who distinguish between conflicts and warnings in matters of implication severity of model changes found in the domain of software engineering. They introduced a matrix for visualising their classification strategy (see figure 3.2).

	Insert	Delete	Update	Use	Move
Insert					
Delete			×	×	×
Update			×	!	!
Use					!
Move					×

×: Conflict

! : Warning

Table 3.2.: Conflict classification by Wieland et al. [Wi13]; Source: [Ro14]

Pursuing slightly different goals as we tread the field of EA model management, ModelGlue implements two more task types: required approvals and validations (introduced in section 3.3.2.1). Furthermore, our concept refines Wieland's approach by amending the dimension of different element abstraction levels. Id est we allow to model different task classifications for different model element types. The case *update object / delete object*, for instance, will trigger an approval task, whereas *update value / delete value* will not. Additionally, ModelGlue enables users to define classification rules across element types, e.g. *move object / create attribute*. In summary, our concept defines five element types (objects, object definitions, attributes, attribute definitions, and values[7]) and five operations (create, move, use, delete, update[8]). Consequently, $(5 * 5)^2 = 625$ possible combinations can be thought of for a detailed conflict classification strategy. Because of the symmetry of operations (*update object / delete attribute* $\equiv$ *delete attribute / update object*), nearly half of the calculated 625 possibilities are redundant. Only $\sum_{i=1}^{25} i = 325$ cases need to be considered.

We offer two different pre-defined conflict classification matrices - 'strict' and 'tolerant'. Figure 3.14 shows the data/data part of the strict classification matrix[9]. Users are free to alter single classifications in the matrix or even specify an entirely new classification strategy. See section 3.5.3 for different conflict classification strategies and section 3.5.4 for custom conflict resolution strategies.

3.5.2.4. The merge algorithm

Figure 3.15 and algorithm 2 (based on [KR14]) illustrate the functionality in more detail.

CollectOperations() retrieves all ChangeSets relevant for the merge process from the input models $M_{1..n}$. It starts with calculating the base time t_b like described in section 3.4.3. t_b plays an important role, since only operations after this time have to be considered. Iterating through all elements of a model, the ChangeSet class provides functionality to query for all ChangeSets of an element that were conducted after t_b. Those ChangeSets can either be provided by the application object cache or have to be retrieved from the data base.

[7]Based on ModelGlue's meta-meta model (see figure 3.3)
[8]Described in section 3.5.1: 'Operation-based approach'
[9]Due to its vast dimensions, the whole matrix can be found in appendix A.

	object update	object delete	object create	object move	object use	attribute update	attribute delete	attribute create	attribute move	attribute use	value update	value delete	value create	value move	value use
object update	conflict														
object delete	approve	none													
object create	conflict	none	conflict												
object move	validate	approve	conflict	conflict											
object use	validate	approve	none	validate	none										
attribute update	none	approve	none	validate	validate	conflict									
attribute delete	none	none	none	none	validate	approve	none								
attribute create	none	approve	none	validate	validate	conflict	none	conflict							
attribute move	none	none	none	none	none	validate	validate	conflict	conflict						
attribute use	none	none	none	none	none	validate	validate	none	none	none					
value update	validate	approve	none	validate	validate	none	approve	none	none	none	conflict				
value delete	none	none	none	none	validate	validate	none	none	none	none	none	none			
value create	none	approve	none	validate	validate	validate	none	none	none	none	none	none	none		
value move	none	none	none	none	none	none	none	none	none	none	none	none	none	none	
value use	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none

Figure 3.14.: Data/data part of ModelGlue's strict conflict classification matrix

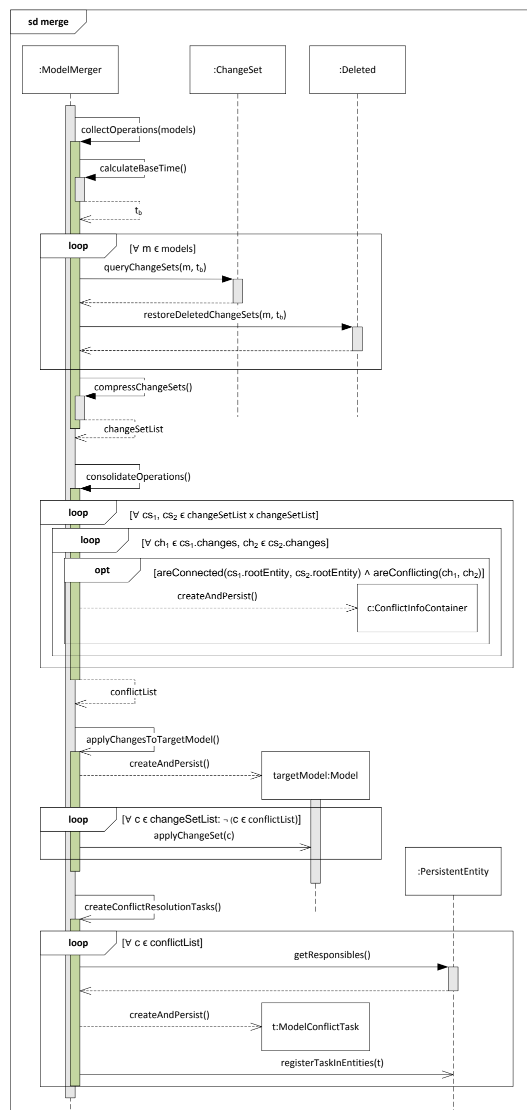


Figure 3.15.: Simplified sequence diagram of the merge algorithm

This way all existing elements can examined. The *delete* operation, however, cannot be detected via this procedure. Tricia moves all deleted entities into a special store in the data base[10]. For detecting delete operations, all potentially relevant elements (Objects and their ChangeSets after t_b) have to be temporarily restored. Before returning the list of all relevant ChangeSets of all given models, insignificant cases are filtered out, thus alleviating the merge complexity. If the same attribute, for instance, is consecutively changed several times, only the most recent change will be kept. This method is called *compressOperations()*.

ConsolidateOperations() analyses the list of relevant ChangeSets and separates between conflicting and clean, i.e. non-conflicting ChangeSets. Since we pursue an operation-based approach when merging models, conflict detection works by comparing operations, divided into ChangeSets and Changes in Tricia vocabulary. Figure 3.11 shows the connection between ChangeSets and Changes. As one ChangeSet might contain several different Changes which could all lead to different conflicts, conflict detection has to be performed on Change and not on ChangeSet level. Iterating through all ChangeSets and all Changes, the heart of the merge algorithm can be structured into three steps:
Firstly, relevant entities are pre-selected by testing whether they are in any way related according to the definition in section 3.4.2 (see 3.5.2.1 for details about step one).
Then, the actual conflict detection determines whether the given changes really cause conflicts in conjunction with the respective model elements. See section 3.5.2.2 for details on this aspect.
Finally, any possible combination of change operations can be looked up in a classification matrix. Section 3.5.2.3 provides a detailed description of this second step. Output of step two is either *null* - then the algorithm continues with the next comparison - or custom conflict resolution rules as introduced in section 3.5.4 or a classification into one of the task types defined in section 3.3.2.1.
In the latter case, a ConflictInfoContainer element is created, aggregating the ChangeSets and Changes currently being investigated. Most importantly, it saves the previously identified classification as meta-information about the conflict.
ConflictInfoContainers are added like shown in algorithm 3. This routine 'appendCIC()' identifies if the newly found situation is part of an already existing case. If that is the case, the new ConflictListEntries are appended to the already existing ones. This mechanism extends the 2-way comparison towards being a n-way merge. When this situation has not been identified before, the new ConflictInfoContainer is simply added to the map.

ApplyChangesToTargetModel() applies all non-conflicting ChangeSets to a preliminary target model M_t, which is also referred to as staging model. At first, M_t is created as the base version M_{t_b} of all input models. This implies that initially the set of ChangeSets in M_t and the set of conflicting ChangeSets found by the method *consolidateOperations()* are disjoint, i.e. that by definition M_{t_b} and hence initially also M_t are without conflicts. Now all clean, i.e. non-conflicting ChangeSets can be applied to M_s. If the user who triggered the merge process specified one of the input models as *preferred model* M_p, conflicting

[10]Note that ChangeSet is a subclass of PersistentEntity. See figure 3.4 for an overview of the class hierarchy.

ChangeSets of M_p are applied as well. However, this will not affect the generation or dissemination of Model Conflict Tasks containing these conflicting ChangeSets. So any conflicting ChangeSets applied to M_t will be marked as such.

CreateConflictResolutionTasks() involves the wrapping of all conflicting ChangeSets detected in the method *consolidateOperations()* into Model Conflict Tasks as well as the dissemination of these tasks to appropriate recipients. One difficulty is to decide who to assign the task to. The simplest case depicted in sequence diagram 3.15 and algorithm 2 retrieves bearers of the responsible role within all elements involved in the conflict and addresses the resulting Model Conflict Task to all of them. This is possible since each ConflictInfoContainer created in *consolidateOperations()* knows all elements involved in the conflict it describes. More advanced assign scenarios are described in section 3.3.2.4. After a Model Conflict Tasks is created, persisted and its addressees are informed, it is finally registered in all elements involved in the respective conflict. Model elements have to know all conflicts they are involved in, as it ensures that constructing model visualisations does not require to iterate through the entirety of Model Conflict Tasks.

3.5.3. Conflict classification strategy

ModelGlue offers two different pre-defined conflict classification strategies. The conflict classification matrix shown in figure 3.14 is referred to as the **'strict' conflict classification strategy**. It comprises many cases where we try to detect potential model inconsistencies and semantic problems. In such cases we trigger validate tasks even though those are no immediate conflicts.

The **'tolerant' conflict classification strategy** only marks those situations as conflicts in which the merge algorithm cannot decide what version to apply. It does not search for semantic impacts or other minor model inconsistencies. This strategy results in the minimal number of conflicts possible.

Both pre-defined classification matrices enable users to alter the classification of every cell. A drop-down list permits to choose between the model conflict task types 'conflict', 'validate' and 'approve' (defined in section 3.3.2.1). If this classification should be to coarse, a fourth 'custom' option can be chosen. See section sec:confStrat for further information about custom conflict handling.

Apart from the two pre-defined classification strategies mentioned above, we offer a third, initially empty, **'custom' strategy matrix**, where users are free to specify their own strategy.

3.5.4. Conflict resolution strategy

If users anticipate typical conflict scenarios for which a general resolution strategy can be specified, they are given the possibility to define **custom conflict resolution rules**. For this purpose conflict classification matrices like the one presented in figure 3.14 provide a

Figure 3.16.: Custom conflict resolution rules interface

'custom' option for every classification cell. If a user selects 'custom' in one of the drop-down lists, he is directed to the user interface shown in figure 3.16. This custom rule dialogue allows to specify the exact conditions the solution applies to and the anticipated conflict solution itself.

3.5.5. Learning and batch-solving

A merge process can potentially result in a considerable number of conflicts. In such a case we try to support the user who solves these model conflicts by trying to learn from the way he solves them. In order to detect recurring patterns in a user's conflict resolution, Schrade [Sc13] implemented the three **learning strategies** shown in figure 3.17. For instance, if a user often chooses values from model A and rarely those from model B as being the correct solution, i.e. if the number of values applied from one model exceed a certain threshold, the learning algorithm suggests to solve all remaining conflict tasks identically. Similar heuristics detect if mostly values being last edited by one certain user are applied. The third implemented learning characteristic detects if a user always applies either the latest or the oldest conflicting versions.

If the learning algorithm identifies a potential solving strategy, i.e. one of the thresholds has been reached, the dialogue illustrated in figure 3.18 asks the user whether all remaining conflicts should be solved likewise. If the user rejects this offer, the corresponding characteristic will henceforth be ignored in this session. If the user should agree to the suggested solving strategy, a **batch-solving** mechanism handles further conflict resolution according to this solving strategy.

Detailed information about learning and batch-solving can be found in Schrade's thesis [Sc13], p. 41ff.

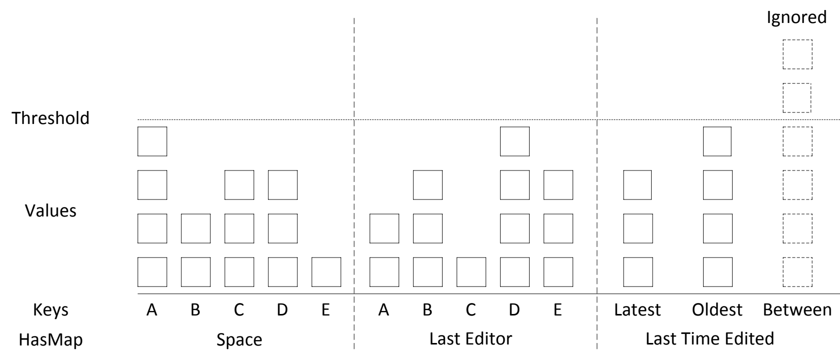


Figure 3.17.: Learning strategies implemented by Schrade [Sc13]

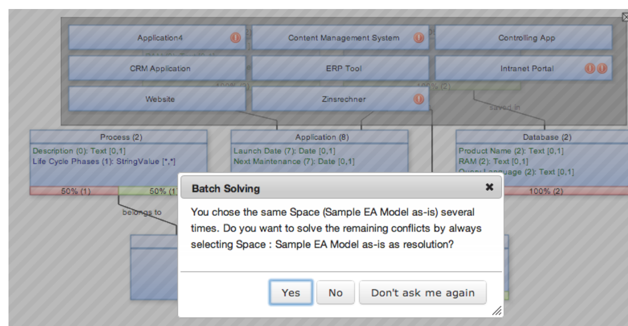


Figure 3.18.: Dialogue suggesting automated batch-solving of further conflicts, based on a strategy learned by analysing user interactions

Algorithm 2: n-way merging of models (evolution of the algorithm in [KR14])

```
   Data: Model M_{1..n}, Baseline Time t_b, Conflict Classification Strategy ς
   Result: Task T_{1..m}, Target Model M_t
1  conflicts, approve, validate ← (key : ∅, value : {∅})
2  customRules ← (key : ∅, value : {∅})
3  changesets ← {∅}

4  // 1. collect operations
5  foreach element ∈ M_1, ..., M_n do
6      foreach changeset ∈ element do
7          if changeset.when > t_b then
8              changesets ← changesets ∪ changeset

9  // 2. consolidate operations
10 foreach cs_1, cs_2 ∈ changesets × changesets : cs_1.rootEntity ⋈ cs_2.rootEntity do
11     foreach ch_1 ∈ cs_1.changes, ch_2 ∈ cs_2.changes do
12         if (cs_1.rootEntity ≇_{ch_1} cs_2.rootEntity) ∨ (cs_1.rootEntity ≇_{ch_2} cs_2.rootEntity) then
13             classification ←
               ς.classMatrix[cs_1.rootEntity.class][ch_1.operation][cs_2.rootEntity.class][ch_2.operation]
14             switch classification do
15                 case conflict
16                     appendCIC(conflicts, new ConflictInfoContainer(/* for meta information */),
                       new ConflictListEntry(cs_1, ch_1), new ConflictListEntry(cs_2, ch_2))
17                 case approve
18                     appendCIC(approve, new ConflictInfoContainer(/* for meta information */), new
                       ConflictListEntry(cs_1, ch_1), new ConflictListEntry(cs_2, ch_2))
19                 case validate
20                     appendCIC(validate, new ConflictInfoContainer(/* for meta information */), new
                       ConflictListEntry(cs_1, ch_1), new ConflictListEntry(cs_2, ch_2))
21                 case custom
22                     customRules.append(cs_1.rootElement, ς.getRuleJSON(custom)
23                 case ∅
24                     continue

25 foreach element ∈ (conflicts.keys ∪ approve.keys ∪ validate.keys ∪ customRules.keys) do
26     foreach cs ∈ changesets do
27         if element ≅ cs.rootEntity then
28             changesets.remove(cs)

29 // 3. apply changes to target model
30 M_t ← M_{t_b}
31 foreach c ∈ changesets do
32     M_t ← M_t ∪ c
33 foreach ruleJSON ∈ customRules.values do
34     ruleJSON.execute()

35 // 4. create conflict resolution tasks
36 foreach conflictInfoContainer ∈ (conflicts.values ∪ approve.values ∪ validate.values) do
37     addressees ← {∅}
38     foreach conflictListEntry ∈ conflictInfoContainer.conflictListEntries do
39         addressees ← addressees ∪ conflictListEntry.changeSet.rootEntity.responsibleRole
40         // See section 3.3.2.4 for further assign scenarios.
41     T_i ← new ModelConflictTask(conflictInfoContainer, addressees)
```

Algorithm 3: Appends new items to the conflict, approve or validate map

```
1  def appendCIC(map, ConflictInfoContainer cic, ConflictListEntry cle_1, ConflictListEntry cle_2):
2      if map.containsKey(cic.baseCLE.changeSet.rootEntity) then
3          cicList ← map.get(cic.baseCLE.changeSet.rootEntity)
4          if (cicList.contains(cic)) then
5              cic.add(cle_1, cle_2) // This conflict is already known. If one of the
                   ConflictListEntries is not stored yet, it will be added. Remember that one
                   ConflictInfoContainer may consist of multiple ConflictListEntries.
6          else
7              cicList.add(cic) // Element already has conflicts, but different ones. Hence, the new
                   ConflictInfoContainer is added to the list.
8          map.put(cic.baseCLE.changeSet.rootEntity, cicList) // Store the updated list.
9      else
10         map.put(cic.baseCLE.changeSet.rootEntity, {cic}) // No conflicts for this element yet.
```

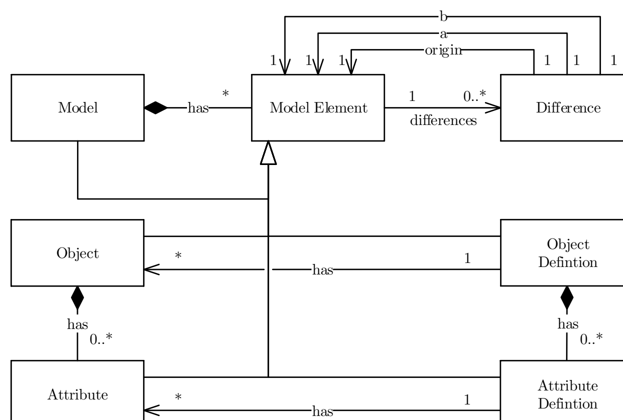


Figure 3.19.: Meta model of the information demand for the difference visualization [RM14]

3.6. Model differencing

Functionality to view differences of EA models plays an essential role in ModelGlue's process for synchronising information sources (see figure 3.1). As demanded by the use cases introduced in section 3.2, difference visualisations should illustrate discrepancies between EA models on schema as well as on instance level.

In contrast to the merge algorithm, the differencing algorithm (also called *Differencer*) takes exactly two models M_A and M_B as input. We always compare between two models and their respective base version, i.e. the last common state, because many users are familiar with visualisations of comparisons between two or at most three objects (e.g. source code, folder structures). Comparing differences between more models would require a greater amount of novel visual concepts. Since we aim at intuitive understanding of visual concepts, differencing is restricted to pairwise comparisons plus one base version.

In contrast to the operation-based approach described in section 3.5.1, no operation or change information is needed, because the differencing routines, originally introduced in [RM14], do not intend to detect conflicts. Therefore the implemented differencing algorithm follows a **state-based approach**.

The two input models M_A and M_B can be seen as static and are thus in line with classical state-based concepts. However, as base versions are not materialised in Tricia, this third model has to be extracted using operations and methods similar to those applied for model merging. Consequently, model comparison conducted in ModelGlue is entirely state-based, but for the integration of the base version also operation-based techniques have to be applied.

Differencing algorithm: Aim of the Differencer is to transform M_A, M_B and differences between these models into a structure apt for serving as input to the visualisation algorithm. Following the design principle of loose coupling [My75], this output structure abstracts from

the complex Tricia input models, striving for a light-weight output model without any unnecessary information. Figure 3.19 illustrates the meta model bridging the gap between Tricia's data model and EA model visualisations. This model is a subset of ModelGlue's meta-meta model. Output of the Differencer depicted in algorithm 4 is a list of $ModelElements$.

The function **compareModels()** in algorithm 4 initiates all comparisons. Since Tricia does not allow hierarchies of *ObjectDefinitions*, they are stored in flat sets. The intersection of both lists of *ObjectDefinitions* contains all tuples of *ObjectDefinitions* t_A and t_B where t_A and t_B are related ($\cong$) according to the rules described in section 3.4.2 ($e_1 \cong e_2 \Leftrightarrow (e_1.uid \equiv e_2.oid) \vee (e_1.oid \equiv e_2.uid) \vee (e_1.oid \equiv e_2.oid)$). *compareObjectDefinitions()* now performs the detailed comparison between t_A and t_B. Any element inside the set union minus the intersection of both lists of *ObjectDefinitions* (line 6) indicates that this *ObjectDefinitions* is either new or has been deleted in one of the models. Nevertheless, those elements without counterparts are still added to the visualisation, as we want to illustrate the entire model.

Objects are stored in a tree data structure. By definition, each tree is an acyclic graph with exactly one root element [Co01]. Consequently, *Object* trees can be traversed recursively, triggered by the call of *compareObjects()* with the two root items as parameters.

The function *compareObjects()* can be divided into the following steps:

1. Line 17: Differences on the level of the two *Objects* are determined. Either o_A or o_B might be new or deleted.
2. Line 18: A $ModelElement$ item is created, which constitutes all information that will later be visualised of the respective *Object*. This element is then added to the global $ModelElement$ list.
3. Line 19: If neither o_A nor o_B equals *null*, the function *compareAttributes()* can now search for differences between the *Attributes* of o_A and o_B. A pointer to the corresponding *Object* allows to add *Attribute* and *Difference* information to the respective $ModelElement$. Since relationships are modelled as attributes with link values in Tricia, the function *compareAttributes()* also deals with associations between $Objects$.
4. Lines 22 to 26: The remainder of *compareObjects()* triggers itself recursively in order to traverse all children of o_A and o_B. Similar to line 6, $Objects$ without related counterpart are added as sole stub $ModelElements$ (line 25) in order for the visualised models to be complete.

compareAttributes() works slightly different than its relatives on higher abstraction levels. As *Attributes* are not visualised as nodes in differences graphs, they are not appended to the $ModelElement$ list but instead attached to the *Object* they belong to. This does not affect difference detection within *Attributes*, though.

The function $createDifference()$ deals with the $Differences$ data structure introduced in figure 3.19 by Roth [RM14]. This routine compares two $ModelElements$. If they are equivalent, no $Difference$ item is created. If only one of the two elements, e_A or e_B, exists but no base version can be found, the respective element apparently is new. If only one of the two elements, e_A or e_B, and a base version exist, a delete operation becomes obvious.

Finally, if both $ModelElements$ exist but are different, they are put side by side with their base version, forming a data structure typical for 3-way-diffs. This data structure is the basis for the fourth layer of differences visualisations.

Algorithm 4: Differencing of models

```
   Data: Model M_A, M_B, Baseline Time t_b
   Result: ModelElement e_{1..n}
1  modelElements ← {∅}

2  def compareModels(M_A, M_B):
3      // Remember the 'related' concept denoted via the symbol ≅ (cf. sec. 3.5.2.1)
4      foreach t_A ∈ M_A.objectDefinitions, t_B ∈ M_B.objectDefinitions : t_A ≅ t_B do
5          compareObjectDefinitions(t_A, t_B)
6      foreach t ∈ (M_A.objectDefinitions Δ≅ M_B.objectDefinitions) do
7          compareObjectDefinitions(t, null)
8          // As we want to visualise entire models, also elements without related (≅) counterpart
              have to be included. These elements are added as stubs. No differences will be
              calculated.
9      compareObjects(M_A.rootItem, M_B.rootItem, null)
10     // Every Model comprises an Object tree with the 'rootItem' being the root node.

11 def compareObjectDefinitions(ObjectDefinition t_A, ObjectDefinition t_B):
12     diff ← createDifference(t_A, t_B)
13     e ← new ObjectDefinition(t_A, diff)
14     compareAttributeDefinitions(t_A, t_B, e) /* similar to compareAttributes(..)          */
15     modelElements ← modelElements ∪ e

16 def compareObjects(Object o_A, Object o_B, Object o_parent):
17     diff ← createDifference(o_A, o_B)
18     e ← new Object(o_A, diff)
19     compareAttributes(o_A, o_B, e)
20     modelElements ← modelElements ∪ e
21     // Object.children is the opposite navigation direction of Object.parent (see fig. 3.3)
22     foreach child_A ∈ o_A.children, child_B ∈ o_B.children : o_A ≅ o_B do
23         compareObjects(child_A, child_B, e) /* traverses the Object tree recursively       */
24     foreach child ∈ (o_A.children Δ≅ o_B.children) do
25         compareObjects(child, null, e)
26         // Element stub without counterpart, similar to line 7. Recursive traversal.

27 def compareAttributes(Object o_A, Object o_B, Object o):
28     foreach a_A ∈ o_A.attributes, a_B ∈ o_B.attributes : a_A ≅ a_B do
29         diff ← createDifference(a_A, a_B)
30         o.addAttribute(new Attribute(a_A, diff))
31     foreach a ∈ (o_A.attributes Δ≅ o_B.attributes) do
32         diff ← createDifference(a, null)
33         o.addAttribute(new Attribute(a, diff))
34         // Similar to lines 7 and 25, we also want to visualise sole attributes.

35 def createDifference(ModelElement e_A, ModelElement e_B):
36     if e_A ≡ e_B then
37         return null
38     e_base ← getBaseVersion(e_A, t_b)
39     if e_base ≡ null ∧ e_A ≡ null then
40         return new Difference<ModelElement>(NEW, e_B, NEW)
41     else if e_base ≡ null ∧ e_B ≡ null then
42         return new Difference<ModelElement>(e_A, NEW, NEW)
43     else if e_base ≠ null ∧ e_A ≡ null then
44         return new Difference<ModelElement>(DELETED, e_B, e_base)
45     else if e_base ≠ null ∧ e_B ≡ null then
46         return new Difference<ModelElement>(e_A, DELETED, e_base)
47     else
48         return new Difference<ModelElement>(e_A, e_B, e_base)
```

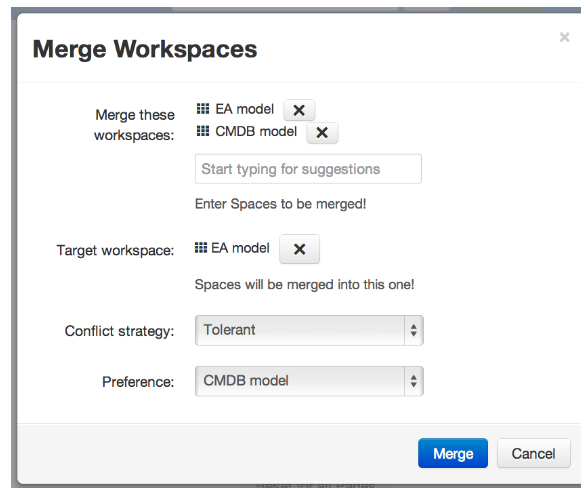


Figure 3.20.: User interface for triggering a model merge process

Meta-information object of merge with Id "114vfqflu4d5x"

Return to merge result table.

Date and time of merge	Mar 12
Conducted by	Enterprise Architect
Merging processed finished	☐
Spaces involved	CMDB model EA model Preview of Merge (114vfqflu4d5x)
Target space	EA model
Preview space	Preview of Merge (114vfqflu4d5x)
Conflict strategy	Tolerant
Preferred space	CMDB model

Figure 3.21.: Interface depicting meta information about a finished merge

3.7. Standard user interface for merge functionality

Starting point of the merge functionality implemented for ModelGlue is the user interface illustrated in figure 3.20. The user is asked to specify the input models $M_1, ..., M_n$ and the target model M_t, whereas M_t is usually one of the input models. Additionally, the merge algorithm may be customised by determining the parameters 'conflict strategy' and 'preference'.

Conflict strategy configures the accurateness and strictness of the conflict detection and classification algorithm. As introduced in section 3.5.2.3, we offer two pre-defined conflict detection and classification heuristics, 'strict' and 'tolerant'. Apart from these two standard strategies, users may also apply a custom strategy matrix.

Preference specifies whether model elements afflicted with conflicts will be applied to the preliminary target model. If the option 'do not prefer any model' is chosen, model elements involved in conflicts will not be applied until a user solves the respective model conflict task and thereby deliberately decides which conflicting version to apply. If, in contrast, one of the input models is determined as preferred model, elements from this model will be applied even if they are afflicted with conflicts. Note that this preference does not affect conflict detection, classification or task dissemination. Model conflict tasks are handled as if no

Date and time of merge	Mar 12
Conducted by	Enterprise Architect
Spaces involved	Architecture Model as-is Architecture Model planned
Target space	Architecture Model planned

Tasks assigned to you:

Page	Status	Related To	Classification	Attribute	Branch A: Architecture Model as-is	Branch B: Architecture Model planned	Base Version	Action
ConflictTask	☐	Web Information Services	UPDATE OBJECT / UPDATE OBJECT Conflict on DATA level	name	Apply: ☐ Internet	Apply: ☑ Web Information Services	Apply: ☐ Application FSF	✓
ConflictTask	☐	CRM Application	UPDATE VALUE / UPDATE VALUE Conflict on DATA level	Next Maintenance	Apply: ☑ 11.02.2015	Apply: ☐ 18.03.2015	Apply: ☐ 11.11.2014	✓
ConflictTask	☐	JBoss Application Server	UPDATE OBJECT / UPDATE OBJECT Conflict on DATA level	name	Apply: ☐ JBoss 7.1	Apply: ☑ JBoss Application Server	Apply: ☐ Application JWJ	✓
ConflictTask	☐	Controlling App	UPDATE VALUE / UPDATE VALUE Conflict on DATA level	Next Maintenance	Apply: ☐ 20.11.2014	Apply: ☐ 08.10.2014	Apply: ☐ 21.10.2014	✓
ConflictTask	☐	Intranet Portal	UPDATE OBJECT / UPDATE OBJECT Conflict on DATA level	name	Apply: ☐ Intranet Portal	Apply: ☐ Intranet	Apply: ☐ Application IVI	✓
ValidateTask	☐	Finance Calculator	UPDATE OBJECT / MOVE OBJECT Conflict on DATA level	name	Revert: ☐ Deprecated Applications	Revert: ☐ Finance Calculator	Revert: ☐ Application SFS	✓

6 rows total — Show 10 rows | 1 of 1

Tasks assigned to you as member of the Administrators group:

Page	Status	Related To	Classification	Attribute	Branch A: Architecture Model as-is	Branch B: Architecture Model planned	Base Version	Action
ApproveTask	☐	IBM z890 mainframe	UPDATE OBJECT / DELETE OBJECT Conflict on DATA level	name	Apply: ☑	Apply: ☐ IBM z890 mainframe	Apply: ☐ IBM z890 (deletedId=zcnhungp9xag)	
ApproveTask	☐	IBM z800 mainframe	DELETE OBJECT / UPDATE OBJECT Conflict on DATA level	name	Apply: ☐	Apply: ☐ IBM z800 mainframe	Apply: ☐	
ConflictTask	☐	Product Life Cycle Management	UPDATE VALUE / UPDATE VALUE Conflict on DATA level	Life Cycle Phases	Apply: ☐ Conception Design Realization Service	Apply: ☑ Conceive Plan Design Realize Service	Apply: ☐ Conceive Design Realize Service	✓

3 rows total — Show 10 rows | 1 of 1

Figure 3.22.: Merge result table

preference had been set, with the difference that the version pertaining to the preferred model is already persisted.
At any time after a merge process is finished, users can recall all relevant meta information via the view shown in figure 3.21.

Merge result table: When the merge process is completed, an overview of resulting model conflicts is presented to the user who triggered the merge (see figure 3.22). This merge result table (also called conflict list) shows all model conflict tasks assigned to the user himself as well as those tasks assigned to roles he currently holds, i.e. groups he is a member of.

Every line in the table illustrates one model conflict task. **Colouring** of the lines visualises the respective model conflict task subtype (see section 3.3.2.1 for an explanation of task types). Red indicates $ConflictTasks$, yellow stands for $ValidateTasks$ and blue visualises $ApproveTasks$. These colours where chosen deliberately, as they denote the different severity of conflict impacts on the resulting model (red is commonly perceived as critical, yellow as medium-critical).

The first five columns of the table show essential **meta information about a conflict**. First the task type, then its status (unsolved/solved), the model element the conflict is related to, and finally a classification and abstraction level of the conflict (e.g. 'update object / move object on data level').

The remaining columns offer means to resolve the conflict directly within this table. The table contains one column for every input model of the merge plus one column for the

Due Date	May 15
Creator	Enterprise Architect
Assigned To	Administrators
Status	☐
Related To	Product Life Cycle Management
Send Email Notifications	☐
Description	Edit
Conflict Classification	UPDATE VALUE / UPDATE VALUE Conflict on DATA level
Conflicting Elements	Product Life Cycle Management in space Architecture Model as-is Product Life Cycle Management in space Architecture Model planned

Conflicting versions:

Object name	Object type	Space	Time	User	Change name	New value	Previous value	Action
Product Life Cycle Management	Page	Base version	27.02.2014 10:58	Max Mustermann	Base version	Conceive Design Realize Service		Apply this version!
Product Life Cycle Management	Page	Architecture Model as-is	18.03.2014 12:10	CMDB Data Owner	HybridPropertyChange of attribute 'Life Cycle Phases'	Conception Design Realization Service	Conceive Design Realize Service	Apply this version!
Product Life Cycle Management	Page	Architecture Model planned	01.03.2014 09:42	Enterprise Architect	HybridPropertyChange of attribute 'Life Cycle Phases'	Conceive Plan Design Realize Service	Conceive Design Realize Service	Apply this version!

Figure 3.23.: Model conflict task details view

base version. Checkboxes enable users to choose which of the conflicting values is correct and hence shall be applied to the preliminary target model. This choice is applied after confirming the resolution by clicking on the 'apply' button on the very right of the table.

Conflict resolution differs slightly for the three model conflict task types. $ConflictTasks$ allow the application of several of the values, hence the checkboxes. When a $ValidateTasks$ is shown, the respective model elements have already been applied. The user can, however, revert one or several of the options. An $ApproveTasks$ forces users to approve of exactly one of the choices, hence the lack of the 'apply' button.

Model conflict task details view: If the meta information illustrated in the merge result table does not suffice to solve a conflict, a mouse click on respective row opens a view on the details of the corresponding model conflict task (see figure 3.23). At the top, this view shows a list of all available meta information of the task, for instance due date and assignees of the task. The table at the bottom details all conflicting versions plus the base version. In contrast to the merge result table, where information about one conflicting version has to be compressed into one table cell, this view dedicates an entire row to every conflicting version. This allows to present further information that might be useful for the resolution of a conflict. For every conflicting version the table prints the user who conducted the change that led to the conflict as well as date and time of this action.

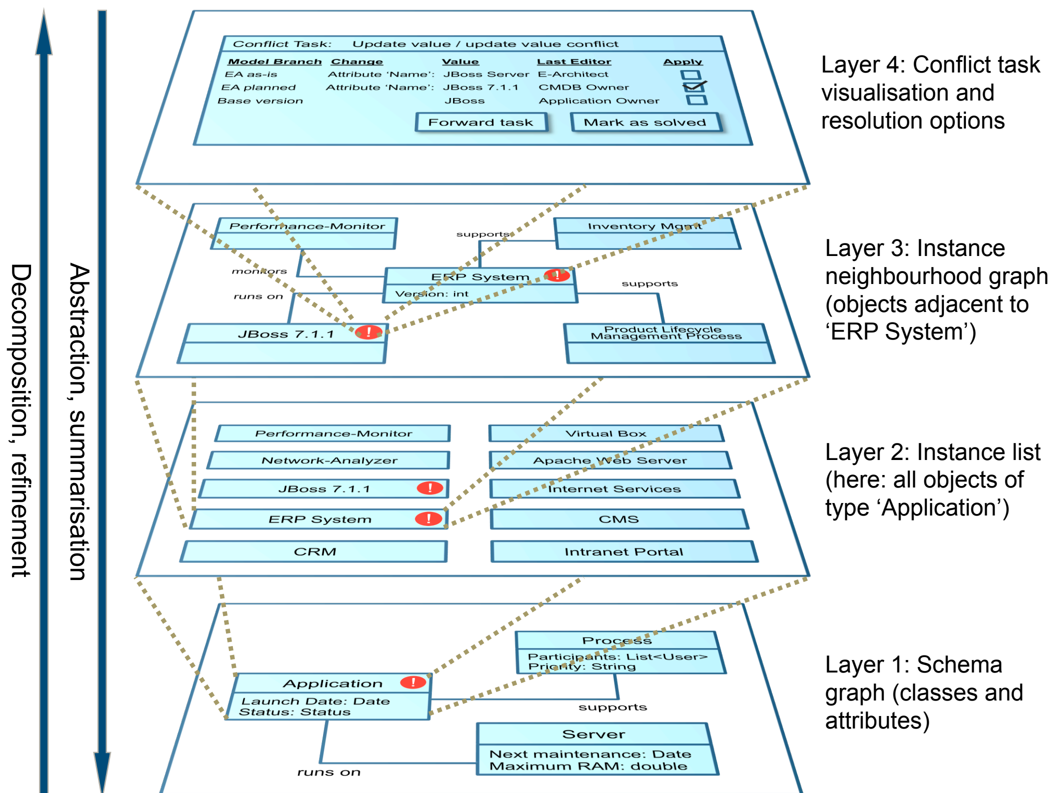


Figure 3.24.: Layers of abstraction in a conflict visualisation of the EA model, following De Marco's principle of recursive decomposition [DM78]. Initial implementation by Tobias Schrade [Sc13].

3.8. Visual concepts

EA models are usually based on "large-scale collections of non-numerical information" [Fr08]. The amount of information contained in these models can make it difficult to grasp the whole picture of the enterprise architecture. Identification of architectural patterns and problems may also become a problem considering huge amounts of raw data. Trying to reduce these problems, visualisations can form a powerful means for understanding EA models [KAV05].

Yet compressing all information of an EA model into a single visualisation probably exceeds the amount of information the human mind can grasp intuitively. So intuitive ways of structuring the information are vital. One of the most effective ways of structuring complex information with the goal of facilitating human comprehension is making use of **hierarchies** [FC93]. In case of EA models, particularly referring to ModelGlues meta-meta model, the meta model part can be seen as inferior to the data part, i.e. schema elements (e.g. object definitions) can can be positioned higher in a hierarchy than data elements like objects. Apart from hierarchies, **modularisation** can help to reduce visualisations to 'cognitively manageable chunks' [Mo09].

As a solution for the mentioned complexity problems, we structure EA models and context information into four layers of abstraction (see figure 3.24). The first three layers visualise

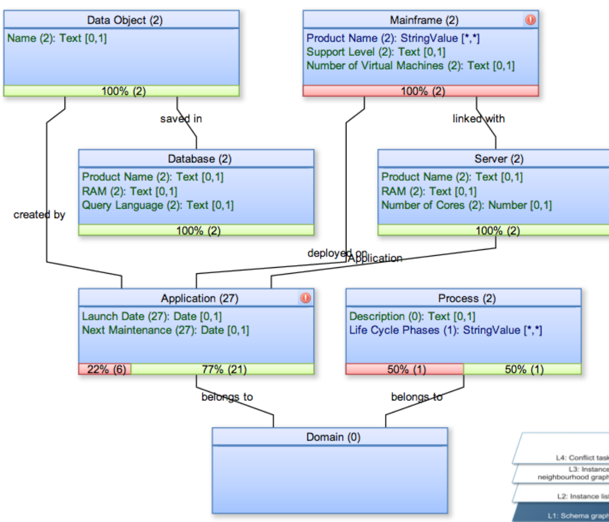


Figure 3.25.: Schema graph (L1) of the conflict resolution dashboard

the EA model on meta model level (layer 1), instance overview level (layer 2) and instance neighbourhood level (layer 3). Note that the structure of these first three layers is identical in the conflict resolution dashboard and difference visualisations. Visual details differ slightly depending of the application scenario (explained in the following sections 3.8.1 and 3.8.2). The fourth layer shows additional information depending on the context of the visualisation. If the visualisation serves as conflict resolution dashboard, the fourth layer illustrates model conflicts, in case of difference visualisations the fourth layer shows a detailed 3-way-diff of the respective model element.

To a certain extent these visualisation layers are similar to the layers in system cartography[11], where IT system maps (a subset of enterprise architecture visualisations) can be enriched by additional visualisation layers depicting exactly the information the user wants to be visualised [Ma08a].

The remainder of this section explains those concepts of layers one, two and three which are common in conflict resolution dashboard and difference visualisations. The following two subsections delineate distinctions and describe the application-scenario-specific fourth layers (conflict task visualisation or 3-way-diff).

Layer 1: Schema graph. Starting point of every EA model visualisation is a schema graph, which illustrates all meta model elements of the EA and relationships between them. Optical appearance of this schema graph resembles UML class diagrams. The classes are *ObjectDefinitions* and their attributes *AttributeDefinitions* in the terminology of ModelGlue's meta-meta model (figure 3.3). Behind the obligatory name of an object definition, the number of objects confirming to this definition is put in brackets. Attribute definitions are also illustrated with a number of instances specifying values for this definition (in brackets after the attribute definition name). Additionally, constraints on the attribute type

[11]A concept introduced by Wittenburg [Wi07]; Formerly called software cartography

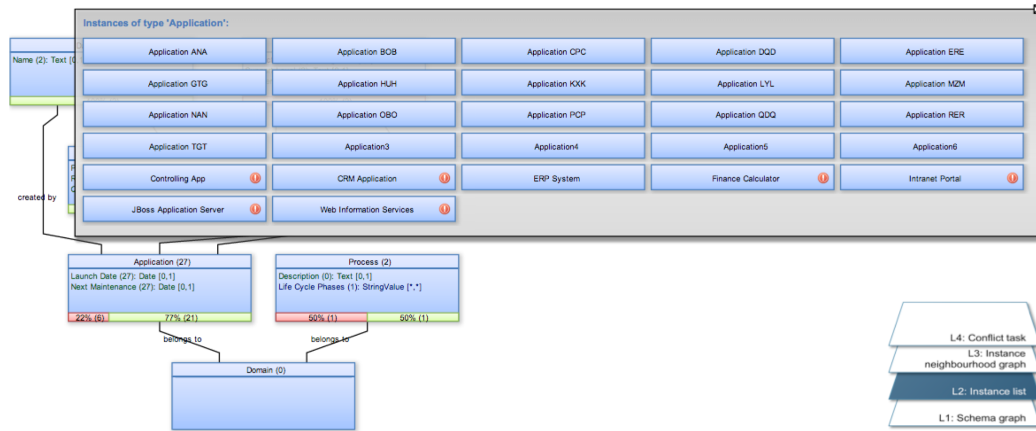


Figure 3.26.: Instance list (L2) of the conflict resolution dashboard

(e.g. 'Text', 'Date', ...) and the required multiplicity ('At most once', 'Exactly once' or 'at least once') are visualised. The concept of constraints on attribute definitions also allows identification and visualisation of relationships between object definitions. Finally, every object definition illustrated in the schema graph has a progress bar at its bottom, giving an overview of how many of the instances of this definition are afflicted with conflicts (in the conflict resolution dashboard) or have differences (in the difference visualisation). This allows a quick insight into the amount of potential problems associated with instances of this definition, even without viewing the instance list.

Layer 2: Instance list. If a user clicks on one of the object definitions (meta model elements) in layer 1, a list of all objects implementing this object definition is shown. Due to the typically very large number of instances ($10^4 < n < 10^5$; see sections 4.1.4 and 4.2.4 for further information about the technical complexity of EA models) the entirety of instances can only be illustrated in a compressed form, i.e. without information about attributes or relationships. If more information about attributes and values of an object are needed, moving the mouse on top of one element hovers a detailed version of this object.

Layer 3: Instance neighbourhood graph. The lack of context information is overcome by layer three. If the user clicks on one of the instances in layer two, ModelGlue opens an instance neighbourhood graph. This graph paints a detailed version of the focal object, which is always allocated in the middle of the window. Around this focal object all its direct relationships to adjacent objects are illustrated together with detailed representations of these adjacent objects.
Moody calls this technique **contextualisation** or 'focus + context' [Mo09]. It shows the conceptual integration of the focal element into the big picture, in this case the entire EA model. By only visualising the immediate neighbourhood, the element's context can be grasped without straining the complexity of the graph. Of course also transitive relationships could be calculated (e.g. relations up to the 2nd, 3rd, or n^{th} degree or even the transitive

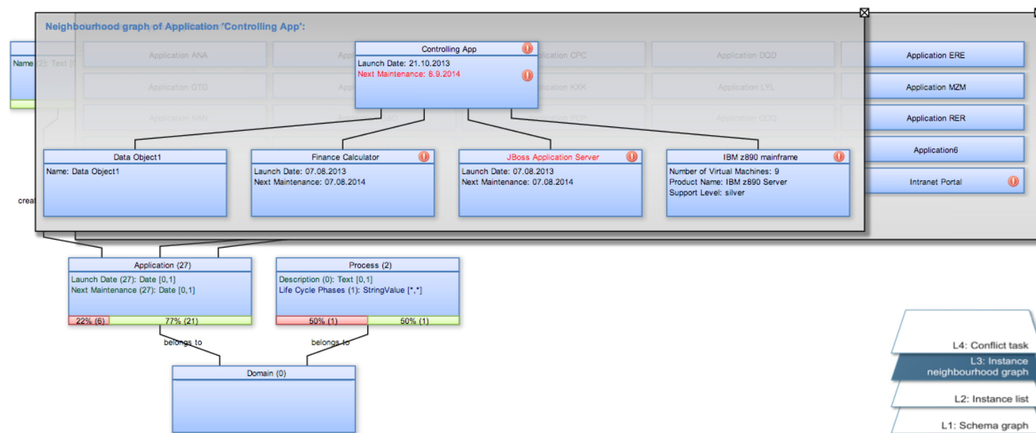


Figure 3.27.: Instance neighbourhood graph (L3) of the conflict resolution dashboard

closure of the entire graph), but considering realistic numbers of instances ($10^4 < n < 10^5$) a larger graph will soon exceed comprehensible dimensions.

Different layers for different stakeholders: Apart from the apparent reduction of visual complexity, the different layers can even address and support different EA stakeholders. The schema graph (L1) could be apt for users incorporating a high abstraction level, like those with a managerial role. Neighbourhood graphs (L3) could support an enterprise architect in his work, and layer four is possibly used by technical users who perform e.g. the actual conflict resolution. Feedback on this aspect can be found in the evaluation part of this thesis (see section 4).

3.8.1. Conflict resolution dashboard

As we pursue the ambition to facilitate conflict resolution via visual means, in addition to the merge result table, model conflicts can be visualised in the context of the whole EA model. Therefore the conceptual structure introduced in the previous section is implemented in so-called **conflict resolution dashboard**. This dashboard visualises the entire EA model in order to provide the necessary context for visualising and solving conflict. On top of this visualisation, the conflicts themselves are presented. Additionally, we offer means to solve model conflict tasks directly inside the visualisation, hence the term conflict 'resolution' dashboard. Initial implementation of the dashboard was done by Tobias Schrade. In-depth information about the technical realisation can be found in his bachelor's thesis [Sc13].

Conflict indicators: In addition to the basic concepts of schema graphs described in the previous section, the conflict dashboard indicates model conflicts afflicted to object definitions by painting a red exclamation mark next to the element name (see figure 3.25). The concept of denoting conflicts via the exclamation mark symbol is used throughout all visualisation levels. In layers two and three (figures 3.26 and 3.27), conflicts associated to objects are visualised similarly to those related to object definitions. If a conflict can be ascribed

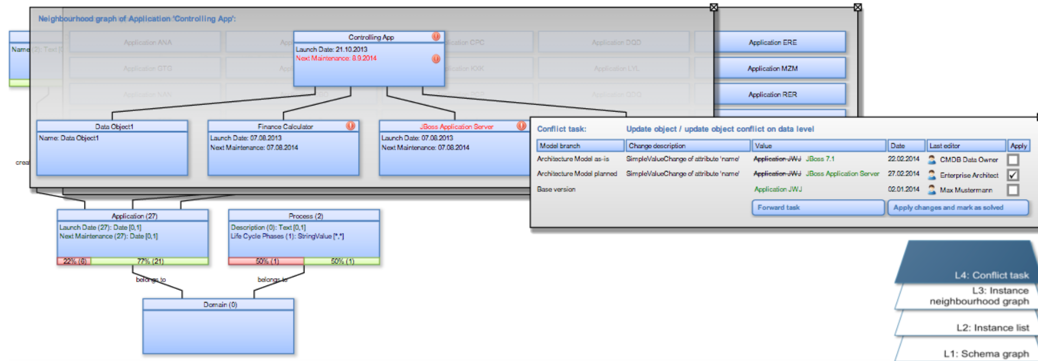


Figure 3.28.: Model conflict task visualisation (L4) of the conflict resolution dashboard

directly to an attribute of an object or an object definition, the exclamation mark is put straight behind the respective attribute or attribute definition. In this case the respective attribute or attribute definition is also written in red colour.

Layer 4: Model conflict task visualisation. A click on one of the conflict indicators (red exclamation marks) opens an overlay with a visualisation of this model conflict task (see figure 3.28). It contains all essential information needed to understand the conflict as well as capabilities to solve the conflict directly in this view.
The headline states the conflict task type (conflict, approve or validate) as well as the classification of the conflict (e.g. update object / move object). The following table visualises all conflicting versions as well as a base version. Every version is shown with important meta information: the model it belongs to, a description of the change that led to the conflict, date of this change and the user who conducted it. Finally, the checkbox at the very right enables a user to choose the correct version he wants to apply. As potentially multiple versions may be applied, these actions eventually have to be confirmed via the button at the bottom. Clicking this button applies all specified changes and changes the state of this conflict task to 'solved'. If the user does not feel responsible for this conflict, he may delegate it via the 'forward' button.

3.8.2. Difference visualisation

The difference visualisation is in many ways similar to the conflict resolution dashboard, only that it does not illustrate model conflicts but differences between two input models M_A and M_B. For indicating model differences we tried to implement an intuitively understandable colouring concept, with different colours for different types of differences (new/deleted/updated). Yet too many colours may confuse users. Steele and Iliinsky recommend not to use more than six colours ([SI11], p. 66). So we ended up with four colours: green for new elements, red for deleted elements, orange for differences within existing concepts and blue as standard background if nothing has changed. This colouring scheme is used throughout all visualisation layers and for all concepts of ModelGlue's meta-meta model including links between elements.

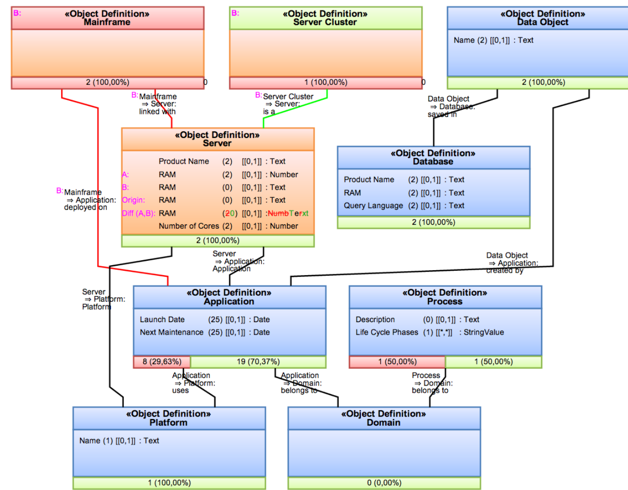


Figure 3.29.: Schema graph (L1) in the difference visualisation

Layer 1: Figure 3.29 shows differences on schema level. In this exemplary scenario an object definition called ‘Mainframe’ has been deleted in model B, while the object definition ‘Server Cluster’ has been created in model B. The orange object definition ‘Server’ indicates differences on attribute definition level. We compare the different versions of attribute definitions by printing a new line for every version (labelled with the key of the respective source model - A, B or *base*) and a further line for a text-based difference view known from source code versioning systems or other text-based diff-tools like kdiff3[12].

Layers 2 and 3: filter functionality is the logical consequence Apart from the colours - green for new objects, red for deleted ones and orange for changed objects - instance list and instance neighbourhood graph do not differ substantially from the same layers in the conflict resolution dashboard. Note that we also detect differences between relationships of objects. In the example illustrated in figure 3.31, the link from ‘IBM z800’ to ‘CRM Application’ has been added in M_A, the link from ‘Front-End Server’ to ‘CRM Application’ in M_B. The red link indicates that it has been deleted in model A. If the user desires more information about an object in layer two (figure 3.30) moving the mouse over this objects hovers a detailed view of the object.

Layer 4: Three way differences view. At the lowest level of abstraction ModelGlue visualises differences between two objects in a further ‘three way differences’ view (shown in figure 3.32). It is called ‘three way’ or also ‘3-way’ because three versions of the focal object are illustrated beside each other. First, the version found in M_A, then the version from M_B and finally their last common state, the ‘origin’ or ‘base version’. This visualisation layer allows quick cognition of differences on object, attribute, and value level. Differences on value level are for instance shown in figure 3.32, where the original value of the attribute ‘Next Maintenance’ used to be ‘1.10.2014’ but has been changed to ‘18.2.2015’ in model B. Differences on object level are illustrated in figure 3.33, where a new application called ‘ERP

[12]http://kdiff3.sourceforge.net

Figure 3.30.: Instance list (L2) in the difference visualisation including detailed element information on hover

Figure 3.31.: Instance neighbourhood graph (L3) in the difference visualisation

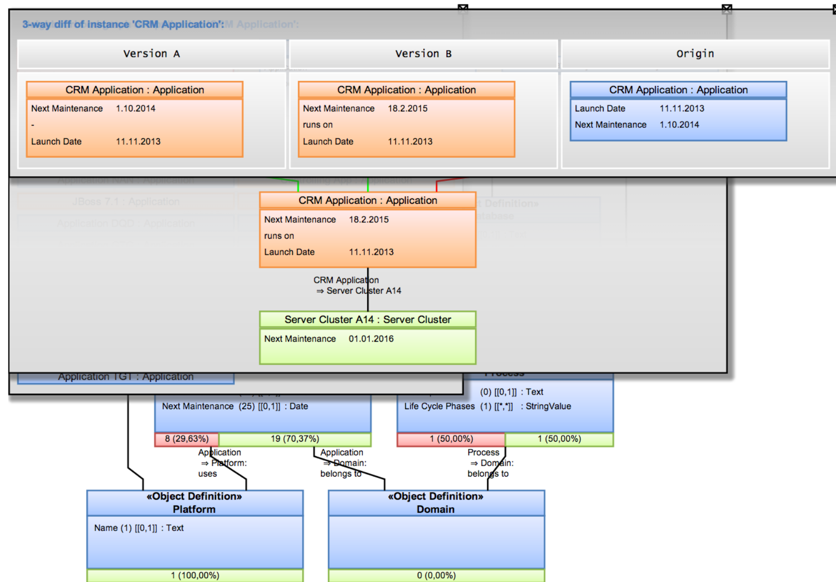


Figure 3.32.: Three way difference view (L4) as overlay on top of the difference visualisation

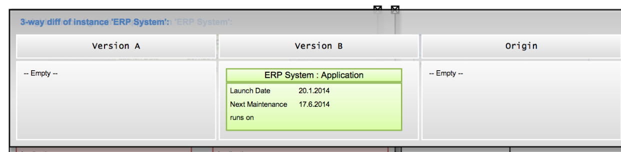


Figure 3.33.: Three way difference view (L4) visualising a new instance

System' is created in model B. Figure 3.34 shows a mainframe server 'IBM z800' having been deleted in model B.

3.8.3. Graphical interaction

As the number of EA model elements being visualised may become tremendously large, users have to be provided with effective and efficient navigation capabilities. Most importantly, navigation includes zoom and pan functionality.
Zoom, i.e. changing the size of the visualisation, can either be achieved by clicking on the buttons with magnifier symbols shown in figure 3.35 or via scrolling the mouse wheel.
Pan, i.e. moving the whole visualisation in order to navigate to the desired spot, can be done via the arrow buttons in figure 3.35 or simply via the arrow keys on the keyboard. A third way to pan is to click onto any white spot inside the visualisation and drag it to the

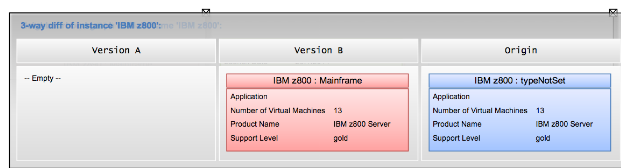


Figure 3.34.: Three way difference view (L4) visualising a deleted instance

desired position.
A detailed report on graphical interaction capabilities implemented within the visualisation framework of ModelGlue can be found in [Ki12].

Figure 3.35.: Graphical interaction functionality provided within visualisations

Apart from navigation functionality, visualisations can also be downloaded as PowerPoint presentation (.pptx file format). This allows permanent storage as well as further manipulation of the downloaded visualisation.

The button with the funnel symbol opens the model element filter described in the following section.

3.8.4. Filter functionality

As we estimated beforehand [RM14] and found out in the evaluation, the whole EA model may easily comprise more than 10^5 model elements. In the case of company A, some object definitions are implemented by up to 24.000 instances. Hence, especially the instance list (layer 2) suffers from the enormous amount of objects. Although our implementation offers effective zoom functionality, visualising all these elements in one graphics eludes human comprehension.

Given this intrinsic complexity of EA models, providing filtering functionality is the logical consequence. Schrade [Sc13] introduced the model element filter shown in figure 3.36. Goal of its design was to offer an user interface which is mighty enough to perform sophisticated queries, but still simple enough to be intuitively understood by EA experts.

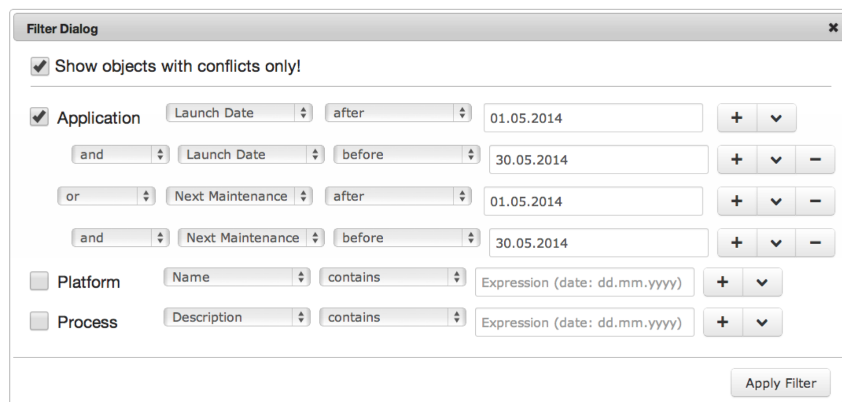


Figure 3.36.: Filter interface

```
{"object definition" : [{
        "conjunction" : "and | or",
        "attribute" : "attribute name",
        "comparator" : "contains | starts with | ends with |
        = | < | > | before | after | not null",
        "predicate" : "value",
        "filter" : [{"conjunction" : "...",
                "filter" : [{"conjunction" : "..."}]
                }]
        },{
        "conjunction" : "and | or",
        "attribute" : "attribute name",
        "comparator" : "contains | starts with | ends with |
        = | < | > | before | after | not null",
        "predicate" : "value",
        "filter" : [{"conjunction" : "...",
                "filter" : [{"conjunction" : "..."}]
                }]
        }]
}
```

Figure 3.37.: Recursive JSON filter for one object definition [RM14]

Primarily, object definitions and all corresponding objects can be filtered out by disabling the checkbox next to their names. The first drop down menu lets users choose one of the attribute definitions of the respective object definition. Next, the second drop down menu specifies the desired logical compare operation (e.g. 'not null', 'contains', '<', '>', ...). Finally, a text field enables to constrain the chosen attribute to a certain value. Such an expression can be amended by further sub-expressions. Sub-expressions can be logically aggregated via 'and' or 'or', or even nested recursively. Figure 3.37 shows the JavaScript Object Notation (JSON) structure the filter is based on. The complete set of supported filtering expressions can be found in [Sc13].

The filter scenario shown in figure 3.36 leads to a visualisation showing only instances of type 'Application' which are launched or have to be maintained in May 2014.

This filter can be used in the conflict resolution dashboard as well as in difference visualisations. Inside the difference visualisation, the option 'show elements with conflicts only' is substituted by the option 'show elements with differences only'.

Filtering via the Model Expression Language

As an alternative to the classical filter user interface described in the previous section, users may write filter queries using the **Model Expression Language (MxL)**. MxL was developed by Thomas Reschenhofer [Re13] at the SEBIS chair as a model-based query language especially for the user-specific definition of key performance indicators (KPIs) in the context of EAM [MRM13].

Similar to modern IDEs (integrated development environment), Tricia's implementation of MxL offers syntax highlighting and code completion facilities. MxL allows the use of many intuitively logical constructs, such as '>' and '<' for comparison of numerical information or 'and' and 'or' for logical joins of sub-expressions. The exact syntax of MxL can be found in [Re13].

```
find(Application)
  .where(("Launch Date" > "01.05.2014" and "Launch Date" < "30.05.2014") or
         ("Next Maintenance" > "01.05.2014" and "Next Maintenance" < "30.05.2014"))
```

Figure 3.38.: Filtering via MxL

```
find (Server)
  .select(
    get Application whereis Deployed.any("Development cost" > 100000))
```

Figure 3.39.: Advanced MxL expression

The MxL query shown in figure 3.38 represents exactly the same filtering example as specified by the classical user interface of figure 3.36. MxL can easily cope with the complexity of simple filtering use cases. However, the big benefit of using MxL is its expressive power. MxL allows advanced queries which exceed the possibilities of the classical filter user interface shown in figure 3.36. An example of such a sophisticated query which cannot be expressed using the classical interface is depicted in figure 3.39. This scenario selects only those 'Server' instances which contain applications deployed on it which have cost less than 100.000 euros[13]. As can be seen, traversing relationships between objects is easily possible via MxL. Thus information about the context of model elements can be used in filter queries, which is not possible employing the classical user interface.

3.9. Collaborative conflict resolution

In order to support collaborative conflict resolution, ModelGlue offers real-time collaboration functionality to review and solve conflicts within the visualization (originally implemented by Schrade [Sc13]). Multiple users can chat, talk, and mutually work on conflicts, similar to the way screen sharing applications work. But as our functionality bases on EA models, different rights and visibilities can be adhered to. In the example shown in figure 3.40, the user with the role 'CMDB data owner' (bottom screen) has no rights to see the schema elements 'Process' and 'Domain'. The user 'Enterprise Architect' (top screen) is allowed to see everything, yet 'Process' and 'Domain' are greyed out, indicating that his communication partner cannot see them.

[13]Corresponding meta model: 'Applications' have a link (called 'Deployed') to 'Servers' as well as an attribute 'Development cost', which contains numerical values.

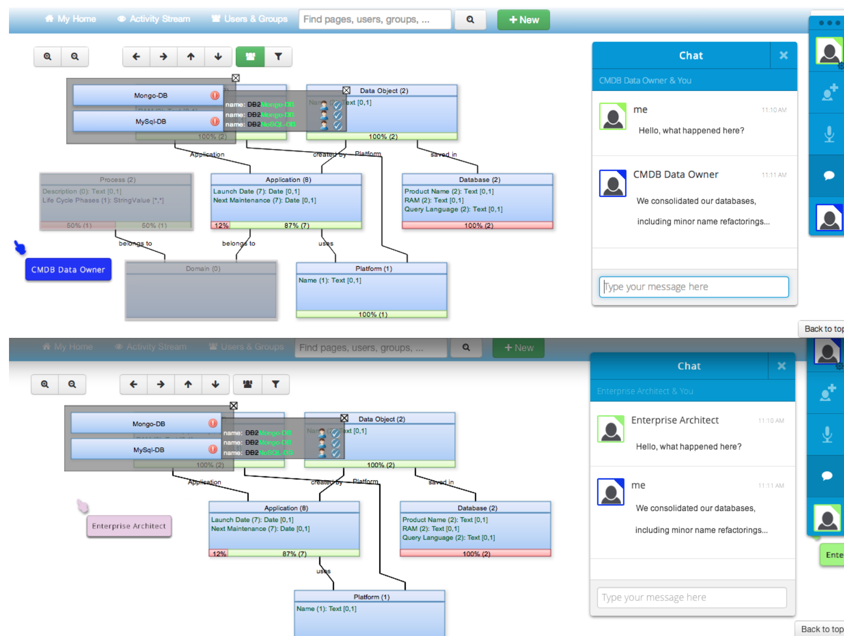


Figure 3.40.: Collaborative conflict resolution

4. Evaluation

Now that the design of ModelGlue has been specified in detail, central aspects of its design have to be evaluated in order to show whether it complies with realistic requirements and user expectations. This section strives to answer the research questions defined in section 2.1, thereby following the research strategy defined in 1.3.

The remainder of this chapter is structured as follows: After a short specification of terminology and abbreviations used in this chapter, section 4.1 deals with evaluation results from case study A, followed by section 4.2 presenting findings from case study B. Section 4.3 compares evaluation results collected from companies A and B. Survey results are presented in section 4.4. Finally, section 4.5 discusses prominent performance characteristics of ModelGlue

Terminology: As stated in section 1.3, feedback in the case studies was collected in several iterations. In every iteration our interview partners provided feedback, pointed out conceptual problems and made suggestions about what could be improved in our design. In order to provide a clear structure and to track feedback, the following sections distinguish between these three kinds of responses.

Feedback: Every chunk of $feedback$ presented in this thesis is given an explicit number and marked with a capital F in calligraphy: $\mathcal{F}_1$, .., $\mathcal{F}_n$.

Problems: Novel conceptual aspects which ModelGlue had not considered before shall be referred to as *problems* $\mathcal{P}_1$, .., $\mathcal{P}_n$.

Change suggestions: Since ModelGlue is a research prototype, this thesis will speak of *change suggestions* instead of the more common term ‘change requests’. These suggestions will be marked with a capital S in calligraphy: $\mathcal{S}_1$, .., $\mathcal{S}_n$.

Further sub-indices denote the company this feedback was collected (e.g. $\mathcal{S}_{A_2}$ or $\mathcal{F}_{B_6}$).

4.1. Evaluation in an insurance company

Case study A was conducted in a large German insurance company (see section 2.2 for a detailed description). As mentioned in 2.3, we conducted three extensive interviews with company A. Table 4.1 gives an overview of our expectations for each of them as well as key findings gained in the interviews. Additionally, company A provided us with a bundle of archival data relevant for their EA models. This iteration will be referred to as D_A. It is listed in addition to the interviews. As stated in section 2.3, interviews two and three were

conducted in a semi-structured way. Questionnaires for these interviews were structured similarly to the survey questionnare in appendix B.

Table 4.1.: Interviews with company A

Interview	Goal	Findings	Sections
D_A	Insights into real-world data	- Data characteristics - Element interdependencies - EA model reconstruction	4.1.2, 4.1.3, 4.1.4
I_{A_1}	Overview of A's EAM and EA model	- A's problems related to EAM - EAM optimisation plans - EA model description	4.1.1 to 4.1.3
I_{A_2}	Feedback on Model-Glue concepts and implementation	- Conceptual feedback - Feedback on ModelGlue behaviour - Assessment of feature relevance - Suggestions for improvements	4.1.3 to 4.1.7 check!
I_{A_3}	Feedback on Model-Glue implementation aspects	- Conceptual feedback - Feedback on ModelGlue behaviour - Assessment of feature relevance - Suggestions for improvements	4.1.6, 4.1.7

4.1.1. Initial state of EAM in company A

Company A, a large insurance company (description in section 2.2), is currently restructuring its EAM. They have already finished the problem identification phase. Interviews I_{A_1} and I_{A_2} revealed profound insights into A's state of the EAM by mid-2013. Current problems A's EAM team identified include the following:

- **Conceptual discrepancy.** One problem consists of conflicts between different levels of abstraction, particularly between highly specialised communities on low abstraction levels and an orthogonal EAM viewpoint on a high abstraction level. Company A also experiences problems with varying interpretation and application of concepts as well as different terminology use. Especially the alignment of business models with those on technical level poses problems.
- **Syntactical discrepancy.** The enterprise is struggling with the diversity of models found across different organisational units. Sometimes even within one department several modelling techniques and tools are employed. E.g. some projects use EPCs (Event-driven Process Chains) for modelling, some use BPMN (Business Process Model and Notation), others do not use any formal language at all. This plethora of technical formats forms an obstacle for any consolidation attempt.
  The lack of shared tool support is an obstacle which of course cannot simply be overcome by a global super-tool. The interaction between the specialised tools of the departments, however, has the potential to run more smoothly. Manual interaction and effort can be reduced.
- **Organisational obstacles.** EAM efforts are often hampered by the multitude of different stakeholders. Even within one community often multiple stakeholders have to

be involved in EA planning, thus leading to slow model coordination and consolidation processes[14].

Goal of the EAM team around our major interview parter P_{A_1} is to align the diverse models and abstraction levels found within the enterprise. They strive to create a cross-departmental view of their enterprise architecture. Thereby, company A expects to realise optimisation benefits.

A's long-term ambition is to improve data quality within EA models in order to support the EAM team with comprehensive, up-to-date information. The closer technical ties between EAM and productive systems shall also lead to a general convergence of architecture and operational communities.

At the start of this thesis in autumn 2013, the described EAM project was still in an early phase of the conceptual alignment and integration of all relevant models.

Technology and concepts applied by company A: For a better understanding of company A's terminology, processes and meta models, it helps to consider basic other concepts they apply. First of all, company A adheres to the ITIL framework (IT infrastructure library), a bundle of recommendations and practices for IT service management (see i.a. [TSO11b]). Especially the terminology used in EA models is often predicated on ITIL concepts.
As a tool for EAM, company A recently introduced Iteraplan [it14]. Iteraplan is based upon the work of Hanschke [Ha12, Ha13a], who distinguishes between the following levels relevant for EA models: Business architecture, information system architecture, technical architecture and infrastructure architecture ([Ha12], p. 55ff). Due to company A's application of Iteraplan concepts, meta models illustrated in this thesis adhere to Hanschke's (and consequently also Iteraplan's) architectural level separation.

4.1.2. Core meta models of the communities

The archival data provided by company A was of essential importance for this case study. It allowed deep insights into data structures of the enterprise.

The data bundle consists of roughly 450 files, representing historical data which had served as an input for A's EA models between March 2009 and July 2013. The bundle is structured into quarterly imports from every information source relevant for A's EA. These time-slices allow the reconstruction of company A's EA model at each of the 16 quarters between Q3 2009 and Q2 2013.

The given data reveals different viewpoints, each of which is relevant to the company's EA model. These viewpoints are incorporated by different organisational units and modelling communities.

- **Product and service management:** This community administers the company's global product and service catalogue (ITIL concept; see [TSO11a]). The catalogue

[14] See [Kh14] for detailed reflections on organisational aspects within federated EA model management.

contains products and services which are offered to internal departments as well as external customers. The team is part of the controlling department, which in turn is an administrative department on the same hierarchical level as the EAM unit.

- **Application development:** This community develops new company-specific applications, chiefly for internal use. It also deals with software change requests requested by other departments.
- **IT asset management:** This community is responsible for the procurement of IT assets, i.e. their acquisition or rental. It also manages IT assets throughout their life cycle.
- **Configuration management database (CMDB):** The community responsible for the CMDB manages the technical IT infrastructure. Its terminology and concepts are based upon recommendations of the Information Technology Infrastructure Library (ITIL) [TSO11b].

Out of the four viewpoints mentioned above we had datasets about the following three at our disposal: *'application development'*, *'IT asset management'* and *'CMDB'*. As it is not directly a part of the EA model, we had no concrete information about *'product and service catalogue'* items. Still, this catalogue is of considerable importance to the EA model, as some key model elements have relationships to catalogue items. Service identifiers are stored as foreign keys in instances of the object definitions 'Application' and 'Product' (see figures 4.1, 4.2 or 4.5).

Data analysis started with separately examining data of the different communities and constructing a schema for each of them. Figures 4.1, 4.2 and 4.3 show the three resulting meta models.

Application development: As mentioned before, 'Applications' are those software solutions developed in-house and for internal use mostly. An application may consist of several subsystems (see figure 4.1). Apart from obvious attributes like name, description, production and replacement date, every application has a status indicating whether it is currently active or not. An application may contribute to a 'Service' listed in the service catalogue. Finally, every application has a responsible person, who can be contacted in case of problems related to it.

IT asset management: Core of the IT asset management are 'Products' (see figure 4.2). Products may be either rented, then the rental has of course begin and end dates, or they may be purchased, then buy date is an important attribute. Both object are associated with the person who requested the purchase or administers the rented product. Products can also be linked to a 'Brand' and a 'Supplier or vendor'. Similar to 'Applications' in the application development community, 'Products' may be part of a 'Service' listed in the service catalogue.

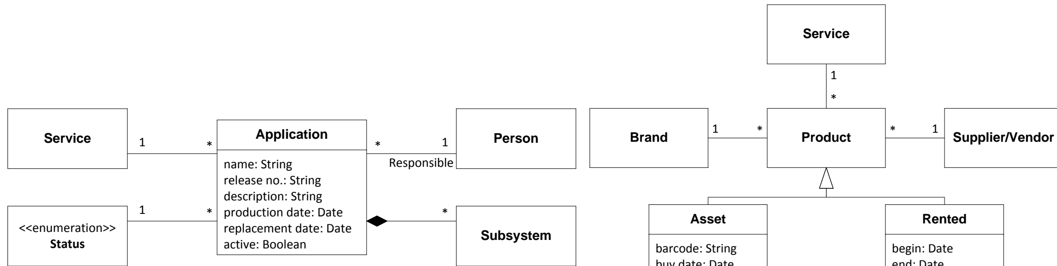


Figure 4.1.: Application development core meta model

Figure 4.2.: IT asset management core meta model

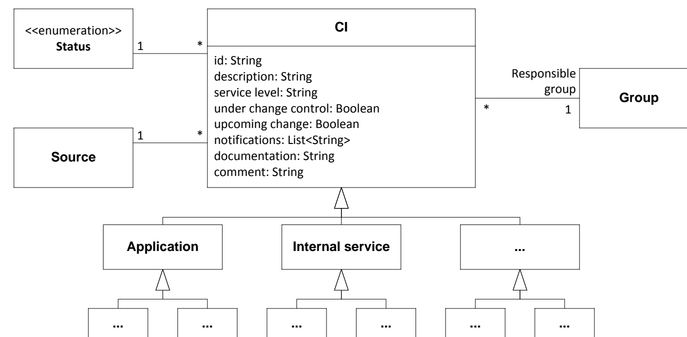


Figure 4.3.: CMDB core meta model

CMDB: A configuration item ('CI') is an ITIL concept for the documentation of low-level entities in the IT infrastructure. Apart from ID, description and a 'Status', it stores its service level and diverse documentation capabilities. Every CI is assigned to a 'Group' that is responsible for this object. Company A distinguishes between several subclasses of 'CI' which in turn have further sub-subclasses. Since the data provided for EAM only contains the two subclasses 'Application' and 'Internal interface', only those were illustrated in figure 4.3 and considered throughout this thesis.

4.1.3. Holistic meta model and abstraction gaps

Knowing that all three of the models shown in the previous section are essential for a holistic EA model, step two consisted of analysing interdependencies between the models and conceptually aggregating them into one comprehensive model.

An early finding was the apparent lack of any direct references between core elements of the three models. To a certain extent 'Application' and 'Product' instances could be mapped via corresponding 'Service' catalogue items. This detour via 'Service' foreign keys is often helpful, however, since one service catalogue item may link to several products and applications, in general it is not unique and thus may lead to ambiguities. Besides, not all applications or products can be associated with a service listed in the global service catalogue.

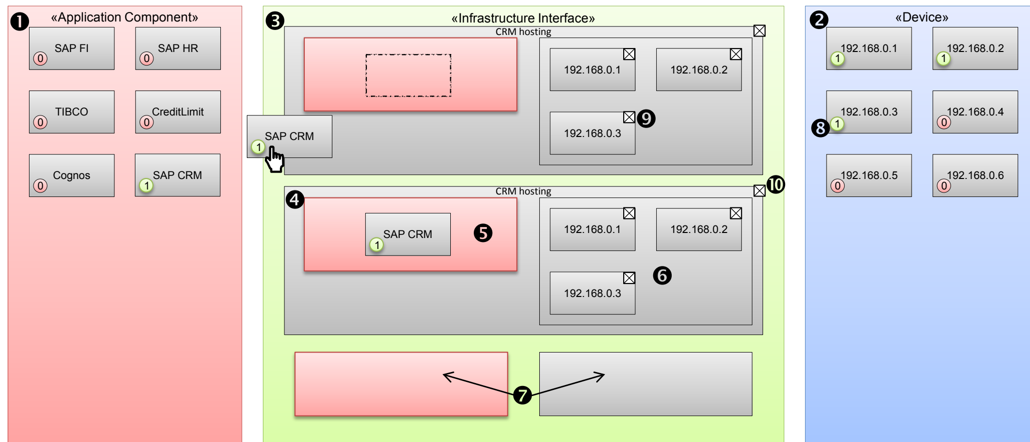


Figure 4.4.: Abstraction gap resolution user interface presented by [RHM13b]

The link between the classes 'Person' and 'Group' was only added for conceptual integrity. This relationship plays no role in the mapping between CMDB and asset management core elements, that is between 'CI' and 'Product' instances.

Such missing ties between model parts which, from a semantic viewpoint, actually belong together are referred to as **abstraction gaps**. Hauder et al. identified abstraction gaps as one of the major and most common problems within automated EA documentation [HMR12]. Roth et al. provide prototypical tool support for overcoming abstraction gaps [RHM13b]. Their interactive visualisation for the resolution of abstraction gaps lets users establish a mapping via drag and drop of model elements. In the example depicted in figure 4.4, a user sees 'Application Components' in a container on the left and 'Devices' on the right. Application components and devices can be linked via 'Infrastructure Interfaces'. Such an interface maps one application to possibly several devices. In the shown example a 'SAP CRM' application is linked to devices with the IPs '192.168.0.1', '192.168.0.2' and '192.168.0.3'. This mapping forms an new instance of type 'Infrastructure Interfaces'. The visualisation provides semantic support for user interactions. In this example the user is currently dragging an object 'SAP CRM'. The visualisation now highlights every possible spot where this object may be dropped in the colour of 'Application Components' (red in this case). The user may thereby alter existing mappings, or, if he drops the object into the red box labelled with 7, create a new mapping[15].

Despite the absence of direct references, analysis revealed possibilities to overcome the abstraction gaps. The most simple way to discern relationships between elements of two communities is to apply a name-matching approach. Ties between 'Application' and 'Product' elements are usually found via exact name matching. Due to two reasons, mapping applications or products to CIs is much harder. Firstly, many CIs may correspond to a single application. And secondly, CIs usually have rather complex names to avoid ambiguity within the CMDB.

[15]Further details can by found in [RHM13b]

Currently, enterprise architects in company A use differencing tools which compare related models on textual file level. Applying the abstraction gap resolver developed by Roth et al. [RHM13b] might alleviate this process.

Peripheral objects: Note that classes with dashed line are references to models of other communities. For instance the class 'Group' shown in the CMDB model is only a foreign key. Groups themselves and information about them are managed by the human resources department, which currently does not contribute detailed information to the EA model. Currently we only integrate indirect information about groups, persons and rights. Consequently, rights and roles in the EA tool differ from those in the source systems. For the application of ModelGlue in real-world circumstances, rights and roles should be imported from a LDAP[16] or similar system.

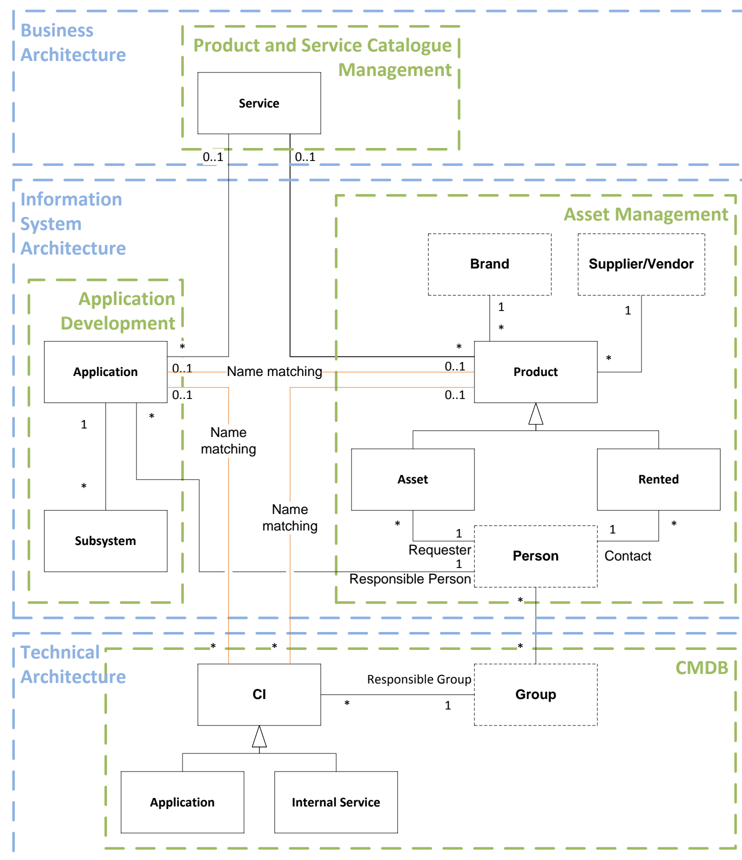


Figure 4.5.: Meta model

Target meta model: As mentioned before, currently company A is engaged in an EAM restructuring process. Their approach towards the alleviation of model consolidation consists of a new conceptual abstraction level between business and IT, called 'application master' (see figure 4.6). This middle layer shall become the new leading system regarding applications throughout the company. It shall be supposed to comprise all applications and relationships between them. Modelling relationships between applications on this level allows to effectively specify constraints. CIs on CMDB level, for example, may only be

[16]Lightweight Directory Access Protocol

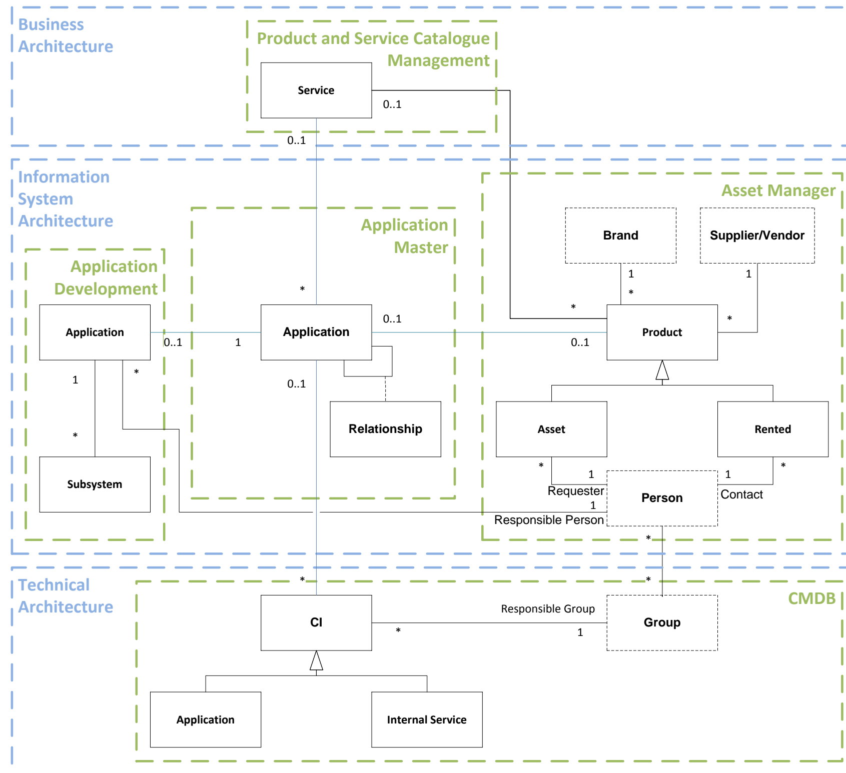


Figure 4.6.: Planned meta model with central application master

connected if their high-abstraction counterparts on application master level share an association. Application Ids generated within the new layer shall be unique throughout the company and serve as foreign key in all other IT-related modelling communities. Thereby, manual name matching is supposed to become obsolete, which enables a higher degree of automation in the domain of EA documentation. Unambiguous identifiers used across all communities essential for EAM also helps to close abstraction gaps.

4.1.4. Technical complexity of the models

Feedback regarding dimension and complexity of all models that are integrated into the EA model gives insights into typical data loads. This information is needed for realistic performance tests (see section 4.5) and finally to answer research question $\boldsymbol{Q_{3_c}}$: 'Does ModelGlue scale considering realistic data loads?'

Scalability can be defined along various dimensions:

a) Persistence level: As ModelGlue is based on Tricia, an established tool also employed by various larger companies, scalability of the technical storage layer is generally not critical. See the performance section 4.5 for further information hereto.

b) Algorithmic complexity: Algorithmic complexity includes e.g. memory and CPU resource consumption of merge and differencing algorithms. This aspect is also examined in the performance section 4.5.

c) Visual complexity: The final, but most apparent aspect. Visual complexity refers to the

amount of information depicted in visualisations and the way it is represented and allocated (via graphs, lists, tables, ...). Goal of ModelGlue's visualisations is to present information in a visual form that is easily comprehensible. As dimensions and complexity of EA models are the primary impact on visual complexity, especially visual complexity will be embraced in this section.

Scalability of ModelGlue's visualisations: For a better understanding of the impact of model size on visualisation size, this paragraph explains two basic mathematical formulas.

$$Number\ of\ sub\text{-}visualisations: \ f(c,i) = \underbrace{1}_{\text{L1 graph}} + \underbrace{c}_{\text{L2 lists}} + \underbrace{i}_{\text{L3 graphs}} + \underbrace{i}_{\text{L4 diff}} \tag{4.1}$$

$$Number\ of\ visual\ elements: \ f(c,i,r) = \underbrace{c}_{\text{L1 graph}} + \underbrace{i*2}_{\text{L2 list+hover}} + \underbrace{i*(r+1)}_{\text{L3 graphs}} + \underbrace{i*3}_{\text{L4 diff}} \tag{4.2}$$

Equation 4.1 shows the number of sub-visualisations depending on the number of object definitions c and the total number of instances i. One visualisation always consists of exactly one schema graph (L1), c instance lists (L2) - one for every object definition - and a neighbourhood graph (L3) and a 3-way-diff (L4) for every instance ($i + i$). Note that the average number of instances per class $\frac{i}{c}$ is important for the perceived visual complexity but not for the actual number of sub-visualisations nor the number of rendered visual elements.

Equation 4.2 calculates the number of visual elements (classes and instances) in a differences visualisation. Besides c and i it takes the average number of relationships per instance r as further input. A schema graph (L1) contains exactly c classes. For every instance, a stripped-down box for the instance list (L2) and a detailed version for the hover functionality is created. Each instance has a neighbourhood graph (L3) showing itself plus its r adjacent instances. If differences to an instance have been detected, (at most) three further visual elements (A, B, base version) are created for the 3-way-diff.

Most importantly, the number of visual elements is a linear function, assuming that c and r are constants or linearly independent from i (viz. $r \neq f(i)$). Linear complexity is an important aspect for scalability. It means that the number of illustrated visual elements is directly proportional to the number of model elements in an EA model. However, even when seen as constants, the model characteristics c, i and r can be really large for EA models and thus imply complex visualisations.

Complexity of core EA models: This paragraph deals with the quantity structure of the analysed EA models and impacts onto their visual representation. It will evaluate early hypotheses about element quantities relevant for ModelGlue's different visualisation layers (published in [RM14]). Roth estimated that in layer one (schema graph) at most 100 classes ($\mathcal{H}_1$), each with up to 10 attributes ($\mathcal{H}_2$), have to be visualised. Layer two (instance list) might contain as much as 10.000 instances of a single object definition ($\mathcal{H}_3$). Finally, layer three (instance neighbourhood graph) should not comprise more than 10 model elements ($\mathcal{H}_4$).

Figures 4.1 4.2 and 4.3 confirm hypothesis $\mathcal{H}_2$, as all EA model elements contain at most

eight attributes. Whenever a user visualises only one of the core meta models, $\mathcal{H}_1$ is always fulfilled, obviously. If an enterprise architect wishes to see the whole joined meta model depicted in figure 4.5, ModelGlue would visualise eleven object definitions[17]. This does slightly exceed $\mathcal{H}_1$'s estimation of at most ten object definitions per visualisation.

Technical complexity of the model around 'Applications':
Of the currently over 1600 'Applications' and 'Subsystems', each one has exactly one link to a 'Service' in the service catalogue. Hence $\mathcal{H}_3$ and $\mathcal{H}_4$ are fulfilled.

Technical complexity of the model around 'Products':
Company A currently manages nearly 3600 'Assets' and more than a hundred 'Rented' products, which is within the scope estimated in $\mathcal{H}_3$. Because not every product is linked to a 'Service' and not every 'Asset' knows its 'Requester', the average product has less than four relationships to be displayed in its neighbourhood graph. At most five visual elements are still well within the estimation of $\mathcal{H}_4$.

Technical complexity of the model around 'CIs':
The configuration management database presently contains more than 22.000 instances. This exceeds the estimation of $\mathcal{H}_3$ considerably. As the entirety of 22.000 instances in one visualisation is not manageable, the filtering functionality introduced in section 3.8.4 is of paramount importance.

Technical complexity of the model around service catalogue items:
As a different team than the EAM community is responsible for the product and service catalogue (see section 4.1.2), the EA team has only rudimentary information about 'Service' instances. However, visualising the 'Service' part inside an EA model allows an illustration of relations to other core model elements (applications and products), which may facilitate impact assessment scenarios.
On average, every 'Service' instance is linked to 18 products and 20 applications and subsystems. Printing $18 + 20 + 1 = 39$ elements in an instance neighbourhood graph exceeds $\mathcal{H}_4$ considerably. Due to the star-shaped allocation of elements in the neighbourhood graph, L3 visualisations with nearly forty elements may become confusing (shown in figure 4.7).

Solution a) Navigation functionality: As described in section 3.8.3 we offer several different means of graphical interaction. Functionality like 'zoom' an 'pan' can help to get an overview of the whole picture and navigate through the visualisation.

Solution b) Filter: The filter introduced in section 3.8.4 of course does not only work on instance list (L2) level but on the entire visualisation and thus also for the instance neighbourhood graph (L3). By deactivating one of the object definitions 'Application' or 'Product' in the filter, the graph complexity can effectively be reduced by half. More advanced filtering scenarios can further alleviate the problem.

Solution c) Reduce amount of visualised information: Currently, the instance neighbourhood graph shows each adjacent instance with all its attributes. But for a quick overview of the instance neighbourhood in L3, object attributes can probably be spared, similar to

[17]Note that Tricia does not allow type hierarchies - otherwise it would be 13 elements.

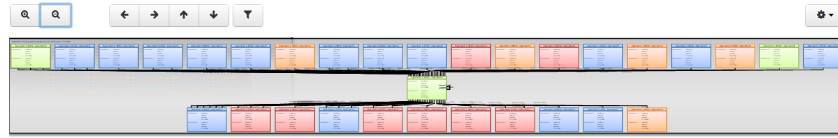


Figure 4.7.: Instance neighbourhood graph (layer 3) with 31 instances

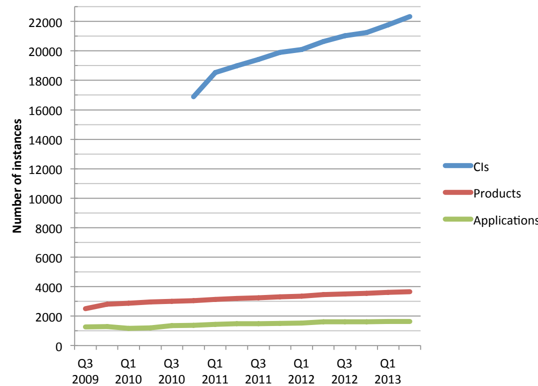


Figure 4.8.: Number of core instances between 2009 and 2013

the instance list in L2. For large neighbourhood graphs, only the focal object should be visualised with all its attributes, whereas the boxes for adjacent objects should only contain name and type of the object. Together with a good graph layout algorithm, this reduction of visualised information can lead to a comprehensible graph, even for a larger number of adjacent instances.

Apart from the visual comprehension problems of extensive graph sizes calculation of a four or even five digit number of visual elements may take a considerable amount of time. Even though we pursue an advanced caching strategy (see section 4.5.2), calculation of such amounts of visual elements and sub-graphs is still time- and resource-consuming.

Model size trend: Figure 4.8 shows the number of instances of important core concepts. The number of applications and subsystems maintained by the application development community grew moderately, but steadily from 1200 instances in 2009 to over 1600 by mid-2013, which is a boost of roughly 30%. Similarly, products modelled by the asset management community saw a constant increase of 47% from 2500 instances in 2009 to 3700 in 2013. As data of CI instances before the fourth quarter of 2010 was not complete, only later values are considered here. From Q4 2010 to mid-2013 the number of configuration items managed in the CMDB increased by 32% from 16.900 to 22.300 instances.
Even without applying statistical methods, a clear trend can be identified. Size and complexity of EA models in company A seem to increase steadily in all modelling communities. Consequently, efficient and effective handling of vast amounts of information in EA models is of vital importance and will remain a challenge in the future, when EA model complexity increases even further.

4.1.5. Application scenarios

This section asks whether the set of application scenarios supported by ModelGlue (see section 3.1) is complete, whether those scenarios are realistic under company A's circumstances and whether they suffice the needs of A's EAM team.

Model-driven planning in the departments ($\mathcal{F}_{A_1}$): ModelGlue supports model management capabilities for a wide variety of possible users. Not only the EAM team necessarily does model-driven planning. Other modelling communities, too, could maintain parts of their information in ModelGlue. This would enable them to run impact scenarios whenever they plan extensive changes. Thereby, the community could discern potential conflicts with data from other communities in advance and avoid conflicts even before the planned changes go live into productive systems.
Feedback: Both enterprise architects of company A appreciated the idea of other communities using such model-driven planning, yet they expressed severe doubts whether anyone apart from the EAM team could be motivated to use this approach. This alleged reluctance can partly be ascribed to the the assessment that many communities do not consider themselves accountable for model impacts onto different viewpoints, but also to the consequence that such model-driven planning would often involve efforts in yet another tool.

Degree of automation ($\mathcal{F}_{A_2}$): ModelGlue strives to automate much of the EA documentation within a company. It taps its full potential when data from all information sources can be integrated via a technical interface or via computer-readable file formats (like e.g. csv).
Feedback: Architect A_1, however, argued that in many cases the matching of data formats deliberately does not get harmonised. Much information still resides within human-readable tools like Microsoft Word, since such tools are easier to use. In many cases this perceived flexibility is favoured over more technical approaches with a higher degree of automation.

4.1.6. Evaluation of ModelGlue concepts

This section presents findings from both interviews I_{A_2} and I_{A_3}, as they complement one another. The first interview provided an overview of the reception of ModelGlue concepts in company A. Interview two showed that some topics required deep discussion and thereby revealed various weak points in the initial ModelGlue design.

Model conflict task types ($\mathcal{F}_{A_3}$): Both enterprise architects from company A agreed to the proposed task types (conflict, approve and validate; introduced in section 3.3.2) as being helpful for weighing different impacts of model inconsistencies.
Discussion about the likelihood of each task type revealed that required validations, as i.a. they detect context changes, will probably be triggered primarily by communities incorporating a rather high abstraction level. Communities on a low abstraction level close to physical hardware are less likely to contain element context changes since hardware hardly alters its purpose. But on an organisation abstraction level, for instance, context changes

might occur rather often when e.g. a development team gets assigned to a new application. Due to the sheer amount of instances and fine-grained details modelled in communities with a low abstraction level like the CMDB unit, these communities are more likely to face many conflict tasks.

Conflict resolution strategy matrix ($\mathcal{F}_{A_4}$): Despite its size and complexity the conflict resolution strategy matrix was intuitively understood by A_1 and A_2. They appreciated the provided flexibility including the possibility to apply more tolerant conflict detection heuristics and thus reduce the number of triggered tasks for certain scenarios.

Long-living conflicts ($\mathcal{P}_{A_1}$): Durable, long-living conflicts may pose a problem for the EAM team. Suppose that integrating data from community C_x leads to a conflict. Ideally, this conflict will get solved immediately. In reality, however, certain situations might impede a quick conflict resolution. Potential reasons for retarding the solution process range from human reluctance to changes of extreme complexity requiring considerable time effort. One example mentioned were 'cleaning projects', where consolidation might even take years, due to few, but substantial changes of the architecture.
Solution: Addressing $\mathcal{P}_1$, ModelGlue was amended by concepts regarding temporal aspects of model conflict tasks. Tasks were given a new 'due date' property. As a consequence, also task states had to be refined. Like shown in figure 3.7, the new sub-state 'overdue' now allows to pinpoint unsolved conflict tasks which have passed the due date. Whenever a task exceeds its 'due date', ModelGlue could send a reminder message. A further idea would be a second escalation step some time after the first reminder. Assuming that all task assignees have left the firm or are on holiday, the second reminder should be sent to the superior of the original assignees, who then are able to re-delegate the task.

Ignored conflicts ($\mathcal{P}_{A_{1b}}$): When conflicts resulting from a merge seem to be irrelevant to the eyes of all stakeholders, users may simply want to ignore these conflicts, without taking the effort needed for conflict resolution. Following the guidance of Matthes et al. [MNS11] such flexibility might foster the modelling process. Similar to the case mentioned above, the task would then live on. However, as ignoring a conflict is a deliberate action, no 'due date' has to be adhered to.
Solution: The obvious solution turned out to be another task sub-state called 'ignored' (see section 3.3.2.2). Model elements that are only connected with ignored tasks are marked with the state 'With ignored tasks only'. Despite them being deliberately ignored, these tasks still represent unsolved conflicts and thus potential problems within the EA model. Data quality of an EA model with many ignored, but unsolved conflicts may be considered inferior to that of the same model where all conflicts are solved.
Due to their impact on data quality of the EA model, ignored conflicts should be illustrated in model visualisations, too. The idea presented in figure 4.9 is to visualise them using the same symbol as for unsolved conflict tasks, but in yellow. Following the popular traffic light colour scheme, yellow is perceived as a less problematic indicator than red.

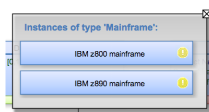


Figure 4.9.: Altered, yellow conflict symbol for indicating ignored conflicts

Quality metrics ($\mathcal{S}_{A_1}$): As stated above, durable and especially ignored conflicts are seen as a flexible solution for quick and simple modelling. They do, however, diminish data quality, one of the goals of company A's EAM endeavour (see section 4.1.1). Hence, A's enterprise architects proposed to introduce quality metrics related to the number of conflicts in a model.
Solution a): A simple yet important measure is the **number of unsolved conflicts** $n_{c/model}$ **in a model**. But due to its absolute character, this metrics is inapt for comparisons between models of unequal size, i.e. with different numbers of model elements.
Solution b): Addressing this comparability problem, a more meaningful measure would be the **average number of unsolved conflicts per model element** $av(n_{c/element})$. One downside of this formula is the fact that it counts all conflicts equally, regardless of their significance.
Solution c): ModelGlue has certain options for estimating the significance of a conflict, particularly along two dimensions. Firstly, conflict impact is affected by the calculated task type (conflict, approve, validate). Exact values are still to be evaluated, but a company could for example define a *task type multiplier* t with $t_c = 1$ for conflict tasks, $t_a = 0,8$ for approve tasks and $t_v = 0,6$ for validate tasks, arguing that conflict and approve tasks have more impact on the EA model than validate tasks. The second input for significance calculation is the state of a task. Deliberately ignored tasks will get a *task state multiplier* s with $s_i = 0,5$ whereas overdue tasks can be weighted with $s_o = 1,5$.
So as a strong metrics we suggest to apply the **weighted average number of conflicts per model element**. av_w is calculated as follows:

$$av_w = \frac{\sum_{i=1}^{no.\ elements} \left(\sum_{j=1}^{no.\ conflicts\ in\ e_i} (t_j * s_j) \right)}{no.\ elements}$$

Recurring conflicts ($\mathcal{P}_{A_2}$): $\mathcal{P}_1$ may raise a further, related problem $\mathcal{P}_2$. Following the scenario introduced in the last paragraph, assume that the conflict of community C_x could not be solved in time. Should the EAM team demand a new import of up-to-date information from C_x before all old conflicts have been settled, exactly the same conflicts as in the last import will be triggered. This conflict redundancy entails some conceptually unsolved questions: Shall the responsible user solve the old conflict first? Dos the old conflict become dispensable? What if the old and the new conflict are solved differently?
Solution: Once identified, this problem could easily be solved on an algorithmic level. Assume that the conflict exists between an element e_{C_x} from community C_x and an element e_{EA} in the EA model. e_{EA} keeps track of all tasks it has been involved in since its creation, and consequently notices that it had been in conflict with e_{C_x} before. If all meta-information

(task type, change operations, ...) about the historic task match those of the new conflict task, the old task will be put to the state 'solved' and a new model conflict task will be triggered.

Learning and batch-solving ($\mathcal{P}_{A_3}$): Schrade [Sc13] introduced learning heuristics for a quicker resolution of multiple conflicts. His algorithms recognise recurring patterns in the way a user solves conflict tasks, e.g. if he always chooses values from model A and never those from model B as being the correct solution. When the number of equally solved tasks exceeds a certain threshold, a dialogue pops up which asks whether all remaining tasks shall be solved identically (called batch-solving; dialogue depicted in figure 3.18). Architect A_1 appreciated this feature as being indeed helpful when many conflicts have to be solved. Still he demurred that to a user's eye the outcome of the batch-solving process is rather vague and not transparent.
Solution: Remedy for this lack of transparency can be provided by a preview of all actions the batch-solving algorithm is about to perform. Ideally, a user could make adjustments inside this simulation view, like e.g. generally approving of the automated solving but excluding certain important model elements which require manual handling.
Different strategies: ModelGlue currently supports three learning strategies implemented by Tobias Schrade[18]: a 'model strategy' (e.g. always apply the element from model X), a 'last editor strategy' (e.g. always take the element where user u_Y has been the last editor) and the 'time of change strategy' (e.g. always take the most recent change). These three strategies could be amended by further heuristics. Enterprise architect A_1 floated the idea of a scenario where community c_X administers purchased applications and community c_Y maintains applications developed in-house. For dependency modelling purposes c_X keeps rudimentary instances resembling elements of c_Y and vice versa. A merge scenario between c_X and c_Y will now inevitably lead to conflicts. These conflicts could, however, be solved by simply applying the rule 'take the version from model c_X if the respective element is a purchased application and the version from c_Y if it resulted from an in-house development'. This rule could be generalised and implemented in an 'advanced origin-dependent strategy'. The example described above is only one suggestion for a new potential learning strategy. More relevant heuristics could enhance the user experience when solving multiple conflicts. As a step beyond pre-defined learning algorithms, A_1 even suggested to offer ways for users to specify custom rules and heuristics matching their personal needs.

EA-specific model alterations ($\mathcal{P}_{A_4}$): ModelGlue supports rule-based conflict resolution strategies as described in section 3.5.4 but had not thought of applying rules even when no conflicts are detected. However, the latter case is deemed important by company A, as it would support the EA team, who often adapt elements to EAM needs. An example would be to add an attribute 'application life cycle' with the possible values 'plan', 'preparation', 'implementation' and 'operation' for every element of type 'application'. The specialised communities that provide the data may not maintain this attribute or use a different value set dictated by the tool they use. Custom rules could also implement automatic value

[18] See section 6.3.5 in [Sc13]

conversion, e.g. 'planning' being mapped to 'plan'.
Solution: With little adjustments, the surface used for custom conflict resolution rules, particularly the MxL interface (see section 3.8.4), should suffice for handling $\mathcal{P}_3$. But due to the limited scope of this thesis it has not been implemented.

4.1.7. Evaluation of the user interface

On a technical level, evaluation of the ModelGlue prototype focused particularly on the implemented EA model visualisations. Section 3.8 provides profound insights into ModelGlue's visual concepts.

4.1.7.1. Differencing visualisation

Concepts and behaviour of differencing visualisations were discussed in interview I_{A_2} ($\mathcal{F}_{A_0}$). Feedback will be structured into the four visualisation layers:

- **Schema graph (L1):** Both enterprise architects of company A affirmed the relevance of this level. Although changes on meta model level were said to be rare, this graph helps to keep an overview of the element structure. This schema graph may prove to be especially useful when comparing models out of different data sources.

- **Instance list (L2):** This layer was seen as a necessary step for keeping track of the complexity in large models.

- **Instance neighbourhood graph (L3):** Architects A_1 and A_2 showed vivid interest in this layer. They stated that the neighbourhood graph could provide great support for impact analyses. Relationships between instances are deemed more important than the mere instance overview provided by L2 including those questions L2 answers (e.g. 'which applications are new?').

- **Instance 3-way comparison (L4):** A_1 and A_2 confirmed that attribute and value changes (e.g. modification of an element description) are the most frequent changes in their EA environment. Consequently, visualising the impact of these changes in a detailed view is generally a good idea. However, according to A_2, attribute and value changes are usually of little importance to the EAM team. The abstraction level of attribute or value changes is often too low for an orthogonal EA viewpoint.
  Suggestion $\mathcal{S}_{A_2}$: On a visual level architect A_1 criticised the background-colouring of the 3-way comparison. Version A used to be painted with a blue background, version B with a green one and the base version in grey (see figure 4.10). These background colours suggest certain semantics - particularly the green one alludes to a new object - even though they have none. Whether an element is new or has been deleted is expressed by the colour of the element box inside the background rectangle. So suggestion $\mathcal{S}_{A_2}$ demands not to use colours where they might lead to ambiguities. $\mathcal{S}_{A_2}$ was seized immediately and realised as shown in figure 4.11.

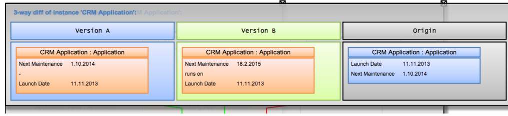


Figure 4.10.: Potentially misleading background colours in the 3-way comparison visualisation (L4)

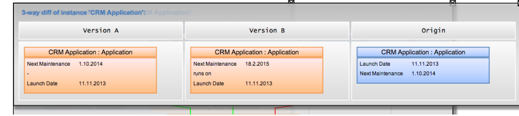


Figure 4.11.: Improvement motivated by company A: unambiguous background colouring

4.1.7.2. Merge visualisation

Feedback on visual concepts related to model merging was collected especially in interview I_{A_3}. As the visualisation layers here follow those of the differencing visualisation, they will not be repeated in depth.

Arrangement of elements in the schema graph ($\mathcal{P}_{A_5}$): ModelGlue uses a third-party library (mxGraph [jg14]) for calculating the layout of all graph visualisations. This library uses algorithms and heuristics which arrange nodes and edges such that they form a compact graph without overlaps. The resulting layout, however, is often perceived as random by end users. According to enterprise architect A_1, users expect elements of a low abstraction level (e.g. the technical infrastructure) to be placed at the bottom of the graph while high level elements (e.g. business processes) should appear at the top. Power users of the tool, like A_1 and other enterprise architects, would cope with the pseudo-random allocation. Occasional users, in contrast, would profit from a logical distribution since grasping the image would be alleviated. Though what is deemed a 'logical distribution' may depend on user groups (e.g. communities) or even on single users. A CMDB user, for instance, sees the world from a different angle than an application developer. Hence, experimenting with the allocation of visual concepts carries great potential.

Constraint violations ($\mathcal{S}_{A_3}$): As described in section 4.2, the enterprise architect of company B proposed to illustrate constraint violations in merge result visualisations. Architect A_1 showed interest for this suggestion but argued that constraint violations represent a different viewpoint than that of EAM. In company A, a separate role than the EA team is responsible for data quality, data documentation and other technical formalities. Generally, this data responsible role would be accountable for constraint violations. In A_1's opinion a data responsible would profit from model visualisations similar to merge and differencing visualisations, only with focus on constraint violations. The EA team, however, does not need this information since it concentrates on model conflicts.

4.1.7.3. Conflict table

Value alterations ($\mathcal{S}_{A_4}$): When solving model conflict tasks inside the conflict table, ModelGlue enables users to choose between all conflicting versions of a model element and the corresponding base version (see section 3.8.1). In addition to this selection capability, architect A_1 motivated a new application scenario. In a conflict between two values v_A and

v_B, the person assigned to solve the task might come up with a different value v_X which he considers more appropriate than v_A and v_B. When for example community c_A maintains a value 'excel2010en_x64' whereas community c_B refers to the same value simply as 'Excel', an architect might want to apply a mixture of the two values, e.g. 'Excel 2010'.
This scenario is only indirectly supported by ModelGlue. The conflict table contains links to the model elements involved in the conflict. Following one of these links, a user can alter the corresponding model element according to his wishes. But adding a value altering feature directly into the conflict table would of course alleviate the procedure and hence be a sensible amendment.

Conflict table versus conflict visualisations ($\mathcal{F}_{A_5}$): ModelGlue offers two coherent yet rather different user interfaces for solving model conflict tasks: the four layered visualisation (see figure 3.24) and a conflict task table (also referred to as conflict list; see figure 3.22) along with task detail views (see figure 3.23). A_1 considers the visualisations as adequate when trying to gain a quick overview of the model and a glance at conflicts involved. It is the ideal means for a manager who wants to identify problems. Afterwards, conflict tasks themselves will usually be solved by a different, not managerial, role, e.g. by a model expert. This model expert is supposedly only responsible for a fragment of the model and perhaps is not even permitted to view the whole model. He will appreciate the compressed information found in the conflict table, particularly if many tasks have to be solved. Summing up, conflict visualisations are advantageous for an overview, especially for managerial roles, while a person assigned to solve a multitude of conflict tasks might rather use the table view for conflict resolution.

4.1.7.4. Conflict resolution strategy

Strategy matrix ($\mathcal{F}_{A_6}$): The conflict resolution strategy matrix was intuitively understood and appreciated by both A_1 and A_2. To assess this intuitive understanding, we only mentioned the general idea as introduction. A_1 and A_2 then guessed the meaning of the illustrated concepts quite right. So we presume that after thorough examination the conflict matrix is comprehensible for enterprise architects.

Custom resolution rules interface: Classical input mask versus MxL dialogue ($\mathcal{F}_{A_7}$): As described in section 3.5.4, the conflict resolution strategy matrix can be extended via custom rules. For this purpose we developed two different types of user interfaces: a classical HTML dialogue inspired by commercial search filter solutions and a MxL-based input. MxL (model expression language) can be used as an advanced query language (see Monahov et al. [MRM13] and Reschenhofer [Re13] for MxL concepts and syntax). An interesting question in this regard is *a*) whether the two solutions are intuitive to the user's eye and *b*) whether their expressive power suffices in the context of EA models ('can the architect express every rule he wants to specify?').
Architect A_1 emphasised the necessity to distinguish between enterprise architects as power-users and other users like data owners, who only occasionally work with the EAM tool.

From a usability viewpoint (a), according to A_1, an enterprise architect would be willing to learn the concepts of MxL and cope with its syntactical complexity rather quickly. Other occasional users, however, probably have little interest in learning a further sparsely used language for 'yet another tool'.
Regarding the necessary expressive power (b), A_1 stated that for simple EA models and basic queries the simple dialogue is adequate. When considering more complex scenarios, e.g. where relationships between adjacent objects or hierarchies are important, the MxL may prove useful and facilitate advanced rule scenarios.

4.2. Evaluation in the health industry

Case study B was conducted in a medium-sized company in the health industry (see section 2.2 for a detailed description). We conducted two personal interviews with enterprise architect *B* from company B. As described in section 2.3, the first interview was mostly unstructured and pursued explorative goals, while the second interview was conducted in a semi-structured manner. Table 4.2 gives an overview of our expectations for each interview and the conclusions we could draw afterwards. In addition to feedback in the interviews, company B provided us with a an excerpt of data used in their EA model. This iteration is referred to as D_B.

Table 4.2.: Interviews with company B

Interview	Goal	Findings	Sections
I_{B_1}	Assessment of ModelGlue process and concepts	- Application potentials of the ModelGlue approach - Conceptual feedback	4.2.1, 4.2.2
D_B	Insights into real-world data	- Data characteristics - Element interdependencies - EA model reconstruction	4.2.4
I_{B_2}	Feedback on ModelGlue concepts and implementation	- Conceptual feedback - Feedback on ModelGlue behaviour - Feedback on visualisations - Assessment of feature relevance - Suggestions for improvements	4.2.2, 4.2.3

4.2.1. Application scenarios

Case study B started with an interview in company B's headquarters. B's enterprise architect was introduced into the general conception of ModelGlue, with focus on the proposed process for federated EA model management (see figure 3.1 in section 3.1) and major high-level concepts involved. Consequently, I_{B_1} concentrated on the assessment of the applicability of ModelGlue in company B's circumstances and the identification of new application scenarios.

After the introduction, architect B explained how he could imagine using ModelGlue and its concepts and where he would make conceptual adjustments. Figure 4.12 shows an annotated version of B's vision of the process proposed by Roth [Ro14] (see section 3.1).

Step 0) (not illustrated): The EA team asks for data from a federated information source A. EA team and stakeholders of source A agree upon conditions for the conceptual and technical data provision. Conceptual questions include the specification of what data is to be provided (selection) and how detailed data can be aggregated in a meaningful way. Technical questions address conditions and format of the export format, for instance csv-files.

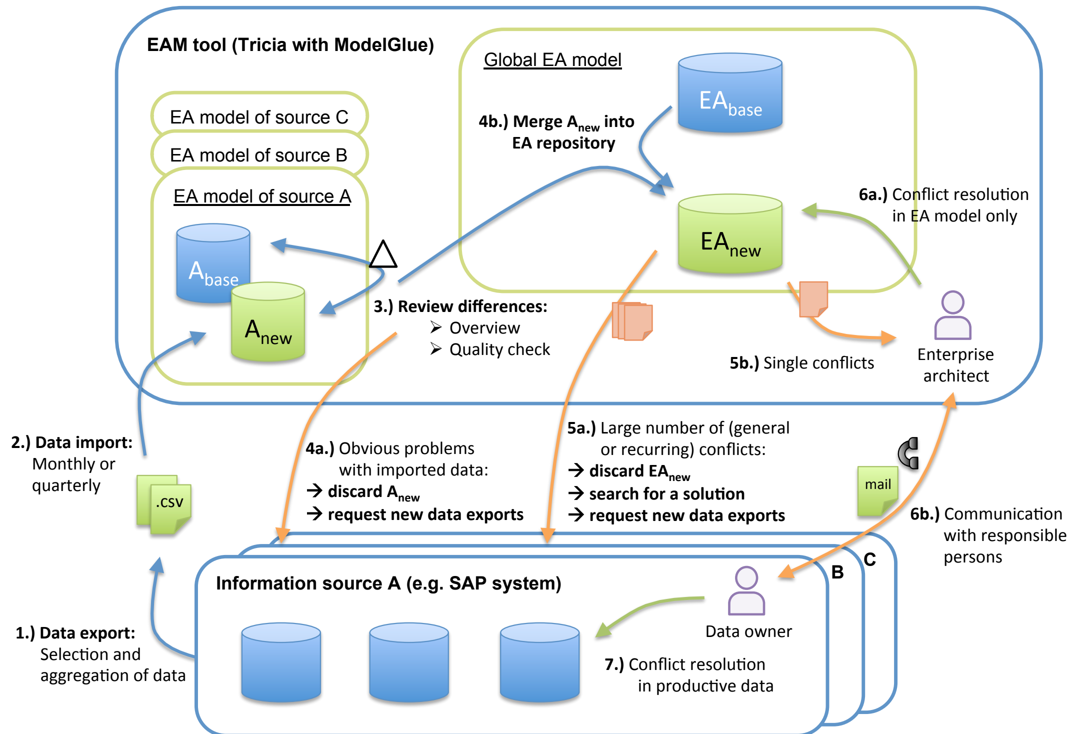


Figure 4.12.: Exemplary application scenario outlined by enterprise architect B

This step corresponds with ModelGlue's 'integrate information source' process step.

Steps 1) and 2): Now that general terms between EA unit and source A are clearly specified, the EA team can request up-to-date data whenever necessary. In company B, such an EA-data-update would usually be demanded every month or quarterly. Export routines care about data selection and aggregation according to the specifications defined in step 0). An enterprise architect subsequently imports the provided data (e.g. in csv format). Architect B calls this a 'semi-automatic' export/import process.

Step 3): An enterprise architect uses the differences visualisation to get an overview of the newly imported changes. Main goal of this step is to detect obvious problems produced by the imported data. Possible problems the architect could identify include the following: technical import failure, missing data, wrong data, data in the wrong format, inappropriately aggregated data. As each import usually updates more than a thousand instances, ModelGlue's visualisation functionality could be a substantial relief for the enterprise architect. Currently, architect B uses Microsoft Excel with custom VBA[19] code for this type of quality assurance.

[19]Visual Basic for Applications

Step 4a): If the data import fails the quality inspection of step 3, the enterprise architect discards the newly imported data, notifies those persons responsible for the information source and especially the data export and asks for a new data export.
Step 4b): If no substantial problems could be identified in step 3, the new data can be merged into the central EA repository, resulting in a preliminary target model.

Step 5a): A merge process leading to an overwhelming number of conflicts (>100) would raise the suspicion that some fundamental reason might be the root of most conflicts. The architect would then desist from solving single conflicts and instead search for this root problem (possibly a problem on schema level or overlapping responsibilities). If need be, stakeholders from information sources involved would have to be consulted and notified of the problem. A potential solution for a responsibility-related problem could be to consider one source as 'master' and the other as 'slave'. After solving the root problem, the whole import process can repeated.

Steps 5b), 6a) and 6b): If the amount of conflicts stays within reasonable limits, the enterprise architect decides whether conflicts are EA-immanent or evoked by their source system or model. In case of the former, the architect himself solves the respective conflicts in the EA model. In case of the latter, the enterprise architect identifies a responsible stakeholder from the source information system, informs him of the problem and discusses feasible solutions. This communication usually happens with little tool support, i.e. typically via e-mail or phone calls.
Step 7): If enterprise architect and stakeholders from the information source have come to a mutual agreement on changes impeding future conflicts in the EA model, a responsible data owner performs necessary changes inside the productive systems. The enterprise architect then alters his EA model accordingly (for small changes) or requests a new data import.

Federated EA model management process - conclusion: Enterprise architect B's vision of a tool-supported process for federated EA model management coincides with most aspects of the process postulated by this thesis. He advocates to solve certain situation detached from a overly strict conflict task regime (step 5a). Because of different tools used in different organisational units without a common basis with the EA tool, B further formulated the belief that other means for conflict task dissemination have to be found. Generally, an enterprise architect would be the person who - via e-mail or phone - communicates problems to the 'right' people responsible for aspects related to productive data.

Application of ModelGlue functionality for planned states of the EA ($\mathcal{S}_{B_1}$): The functionality bundle offered by ModelGlue allows to branch parts of the EA model, envision planned states of the EA, and simulate expected impact on other parts of the EA. Since

ModelGlue always treats data and schema (meta model) as a whole, such planned-state scenarios can be performed on both data and schema abstraction level.
Architect B embraced the general idea, especially on schema level, yet expressed doubts about whether the data part is relevant in planned-state scenarios.
Solution: A solution might be to decouple the schema from its data whenever that is desired for certain scenarios. ModelGlue's branching functionality (see section 3.4) can be equipped with a parameter 'branch schema only'. If a user sets this flag, only meta model elements get copied into the new model branch. This action would not negatively affect any other functionality provided by ModelGlue. Its flexibility allows users to still use differencing and merge capabilities regardless of whether a schema contains data or not.

Visual (meta) model manipulation functionality for planned-state scenarios ($\mathcal{S}_{B_2}$): When modelling planned states of an EA model, meta model visualisations are of vital importance (see also $\mathcal{S}_{B_1}$) both for the action of modelling changes to the architecture and the visualisation and interpretation of resulting impacts. Yet currently, ModelGlue only offers the latter functionality, i.e. impact calculations (via model merging) and visualisations of EA models. The actual modelling of changes cannot be performed inside these visualisations. As most EA tools offer visual model manipulation capabilities, users would expect such functionality from ModelGlue, too.
Solution: Apart from basic features like pan and zoom, ModelGlue currently does not offer visual model manipulation functionality. Its underlying framework, however, was designed to support visual interaction. Basic concepts of the visual interaction framework were developed by Schaub et al. [SMR12]; an advanced visual interaction example can be found in [Ki12]. Consequently, implementing further visual capabilities follows ModelGlue's conceptual roots and would not pose substantial problems.

4.2.2. Evaluation of ModelGlue concepts

This section takes a closer look at selected ModelGlue concepts. $\mathcal{F}_{B_1}$, $\mathcal{F}_{B_2}$, $\mathcal{P}_{B_1}$, $\mathcal{P}_{B_2}$ and $\mathcal{P}_{B_3}$ were discussed in interview I_{B_1}.

Coverage of a central EA model ($\mathcal{F}_{B_1}$): B emphasised to restrict the coverage of a central EA model to a small scope. Fine details alone are of little importance for the high-level view the enterprise architecture has to provide. The same details may, however, provide substantial benefits when cumulated into meaningful information. Such aggregated information facilitates gaining a quick overview. Architect B pictured an example of information from the accounting community: In-depth data about under what terms a product is paid for, if it is e.g. paid monthly or per year, is irrelevant considering the viewpoint of EAM. Yet the very same information in a cumulated form could serve EAM.
This thesis does not intend to give advice about what to model in EAM. Nevertheless, feedback $\mathcal{F}_{B_1}$ is laid out here, as it does have implications for the technical complexity of EA models.

Technical integration of information sources ($\mathcal{F}_{B_2}$): Architect B advocates a light-weight approach for the technical integration of information sources. Interfaces for the data import, i.e. via REST (representational state transfer; see [Fi00] p. 76ff) web services, are useful but should not be over-specified. Since he argues for a semi-automatic or even ad hoc import process, even the exchange of data via csv-files can be sufficient.
Implementation in ModelGlue: ModelGlue currently contains a prototypical implementation of csv-file import mechanisms trying to offer sufficient flexibility to cope with diverse input formats. Yet since actual data import functionality is highly company-specific, this matter is deliberately paid little attention in this thesis.

Unidirectional information flow ($\mathcal{P}_{B_1}$): According to B, information can only flow unidirectionally, from an information source to the EA tool - never the other way around. Implementing bidirectional interfaces would be too complex. Such a deep integration of information sources could also lead to security problems, e.g. when live productive data is manipulated.
Information flow in ModelGlue: ModelGlue follows this principle demanded by architect B in matters pertaining to the direct manipulation of productive data. An enterprise architect is not meant to alter data in primary information sources. ModelGlue does, however, strive to alleviate the information flow back to the sources on a communicational abstraction level. That is, the concept of model conflict tasks pursues the vision that meta-information about conflicts can be disseminated throughout the entire company.
Summing up, ModelGlue separates information flow into **a) data flow** into the EAM system (unidirectional) and **b) meta-information flow** (tasks) which can be applied bidirectionally.

Model conflict occurrence ($\mathcal{P}_{B_2}$): Enterprise architect B is sceptical about the significance of model conflicts in a typical information source synchronisation scenario. He argues that if a merge process results in more than thirty or even in a three-digit number of conflicts, some conceptual mistake or technical error must have slipped in. In such a case an enterprise architect would not solve every single conflict but instead search for conceptual roots of the problem. Changing the schema of the imported data could be the key to prevent the majority of the conflicts.
Solution: ModelGlue supports this approach since all merge results are saved as preliminary models (see section 3.5.2). These preview models can easily be discarded in case of severe problems. After an enterprise architect has identified and solved the root problem of most conflicts, the data import can be repeated.

Model conflict task dissemination ($\mathcal{P}_{B_3}$): *Problem:* Lack of a common platform for model conflict task dissemination.
In company B, Tricia is only used by the EAM team and several more organisational units. Hence assigning model conflict tasks in ModelGlue to people who hardly ever use the tool makes little sense. Architect B points out that in the case of his enterprise, where only very few people are entrusted with maintaining the EA and the EA tool is not part of a technical

platform used throughout the entire company, advanced conflict task assign scenarios are dispensable. The enterprise architect who triggered the merge would be the addressee of all resulting conflict tasks. If a conflict indicates problems which have to be solved directly in one of the primary information sources, the architect would phone or e-mail the respective community.
Solution a): Tricia's task implementation allows to send notifications via e-mail. Yet these mails currently only act as a reminder with a link to the respective task in Tricia. Users who do not use this tool can hardly be encouraged follow a link into an alien system. Hence an enhancement would be to enrich computer-generated e-mails with enough information to be able to serve as the sole source of information needed to solve the conflict. Such an e-mail should contain all conflict meta-information in a structured format. A mail representation of the conflict could look similar to the conflict visualisations proposed in this thesis (see section 3.8). This approach could be the method of choice for all users who do not use Tricia - or generally speaking for all cases where the EA tool is not widely used throughout the enterprise. Downside of this solution is that in some cases it may lead to a flood of e-mails to a single person, which could cause resentment against the EAM tool.
Solution b): Another possible approach would be to integrate model conflict tasks into the company's intranet portal, depending on how frequently it is used. In any case, tasks should be distributed in a technical format which is accepted and frequently used by a great majority of the potential task addressees, be it a specialised EA tool or an intranet portal or simply e-mail communication. Due to the loose coupling between visualisation functionality and the underlying data layer, ModelGlue's conflict visualisations can easily be detached from Tricia and integrated into different tools.

4.2.3. Evaluation of ModelGlue visualisations

In interview I_{B_2} the actual implementation including all visualisations was shown to the enterprise architect from company B. Accordingly, this interview focussed on low-level aspects of ModelGlue's concepts.

Legend for visual concepts ($\mathcal{S}_{B_3}$): As ModelGlue's visualisations apply a plethora of visual concepts, a legend explaining these concepts would facilitate quick comprehension of the visualisations, especially for novel users.

Transparency of visual overlays ($\mathcal{S}_{B_4}$): Architect B criticised the high transparency of visual overlays. Elements from deeper visualisation levels could be confused with those of a higher level. Moreover, in some cases overlay transparency made it difficult to spot the focal visualisation level, i.e. the highest, currently active level.
Solution: $\mathcal{S}_{B_4}$ was seized and led to the improvement shown in figure 4.14. A drop shadow helps to emphasise the separation of different visualisation layers. In combination with other details, this improvement led to a more friendly and clean appearance.

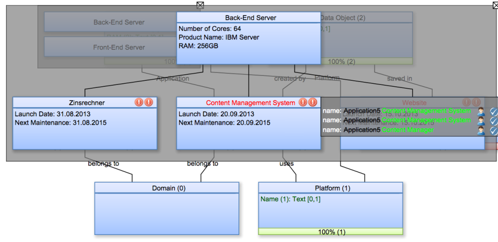


Figure 4.13.: Visual overlays with highly transparent background

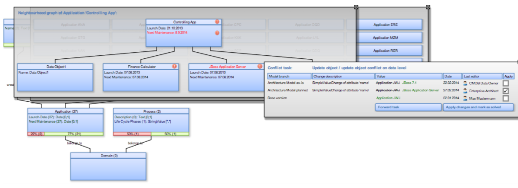


Figure 4.14.: Improved overlay background including shadows to emphasise layer separation

Intuitive comprehension and expressiveness of the filter ($\mathcal{F}_{B_3}$): The filtering user interface presented in this thesis (see figure 3.36; originally introduced by Schrade [Sc13], p. 27f and p. 48ff) was developed with the clear aspiration of being self-explanatory. To test this pretension, architect B was confronted with the filter view without any explanation of the functionality. After a thorough look he confirmed that the filter was intuitively understandable by someone who works as an enterprise architect or on a similar level.
A further question was whether the filter would suffice to express all queries typically needed in the field of EAM. As he could not imagine counter-example, architect B affirmed that the expressiveness of the filter was adequate for EA model management purposes.

Filter improvement suggestions ($\mathcal{S}_{B_5}$): Despite his generally positive reception of the filter, enterprise architect B expressed a few suggestions for further improvement.

Adapt filter for constraint violations ($\mathcal{S}_{B_5} - a$):
The first suggestion architect B expressed upon viewing the filter interface was to include the possibility to filter also for constraint violations. Constraint violations could for instance be missing compulsory values. They may be seen as quality indicators for model elements. Thus validating the EA model via highlighting constraint violations allows conclusions about quality characteristics of the model. This suggestion follows B's general vision to use merge and differences visualisations for quality assurance (see section 4.2.1).

Filter queries over all object definitions ($\mathcal{S}_{B_5} - b$):
Originally, ModelGlue's element filter was designed to only define queries on one (or several) selected object definitions, but never across object definitions. Filtering for all instances containing 'SAP' in its name regardless of the instance's object definition is not possible in the current implementation. For this exemplary scenario the user would have to repeat the 'contains' attribute sub-filter for every object definition where he believes 'SAP'-related instances might occur. Since repeatedly entering equal expressions is inconvenient, architect B would appreciate the possibility to define cross-object-definition filter expressions.

Decouple filter from visualisations ($\mathcal{S}_{B_5} - c$):
As an enterprise architect does not always work inside visualisations of his EA models, but still on the same data, he may also want to use a mighty model element filter outside of the visualisations. Consequently, providing filtering functionality as presented in this theses but detached from model visualisations would also be appreciated by architect B.

Save filter queries ($\mathcal{S}_{B_5} - d$):
Architect B finally demanded that it would be helpful if the filter was reproducible. Users would most likely apply the same filter several times when working with a visualisation for a longer period of time. Since repeated insertion of an advanced filter scenario can be rather tedious, he asks for functionality to save filter configurations.
Solution: For internal representation and transportation from client (web browser) to server, the filter is already stored in JSON (JavaScript Object Notation; see section 3.37 or [RM14] for details about the JSON representation). As the JSON standard allows easy serialisation and storage, implementing a new 'save' functionality for the filter does not pose technical problems.

Comprehensibility of MxL expressions for filtering ($\mathcal{P}_{B_4}$): Filter queries using the Model Expression Language (MxL) as defined in section 3.8.4 were perceived as being not particularly user-friendly by enterprise architect B. He identified the following problems:
Problem a): Most of all, few people in company B are really engaged in EAM. Hence, apart from architect B himself there are no real power-users for the EAM tool. The motivation for users to learn a sophisticated language like MxL is rather low when they would only sporadically come into situations where the language can be applied.
Problem b): Additionally, MxL is not a widespread language. As a consequence, finding help for potential problems could prove difficult. Tricia does, of course, contain documentation about the language, yet architect B would wish for a vivid web community that could answer novel problems before really using MxL.

Merge preference ($\mathcal{F}_{B_4}$): After being presented the user interface for triggering model merges (see figure 3.20), architect B emphasised the relevance of specifying a 'preferred model' which has priority over all other models when merging EA models. Specifying the 'preferred model' option can lead to a substantial decrease in detected model conflicts. B recommends a clear separation of responsibilities, i.e. to declare one information source as 'master' and all others as 'slaves' in regards of shared model elements. Following the master/slave principle, he estimates that 90% of the newly imported changes will be applied from the 'master' model, which corresponds to the 'preferred model' concept presented in ModelGlue.

Conflict progress bar ($\mathcal{F}_{B_5}$): Enterprise architect B welcomed the bar beneath each object definition in schema graphs (see e.g. 3.25), which shows how many instances of one object definition are afflicted with conflicts. The bar facilitates a quick detection of conflict 'hot spots' and guides the user's attention towards critical parts of the EA model. This fosters an easy model quality assessment and conflict identification.

Base version ($\mathcal{P}_{B_5}$): Architect B doubted the necessity for visualising the base version as additional option in conflict tasks as well as in layer four of the difference visualisation. He stated that usually a two-way-comparison would suffice in the context of EA models.

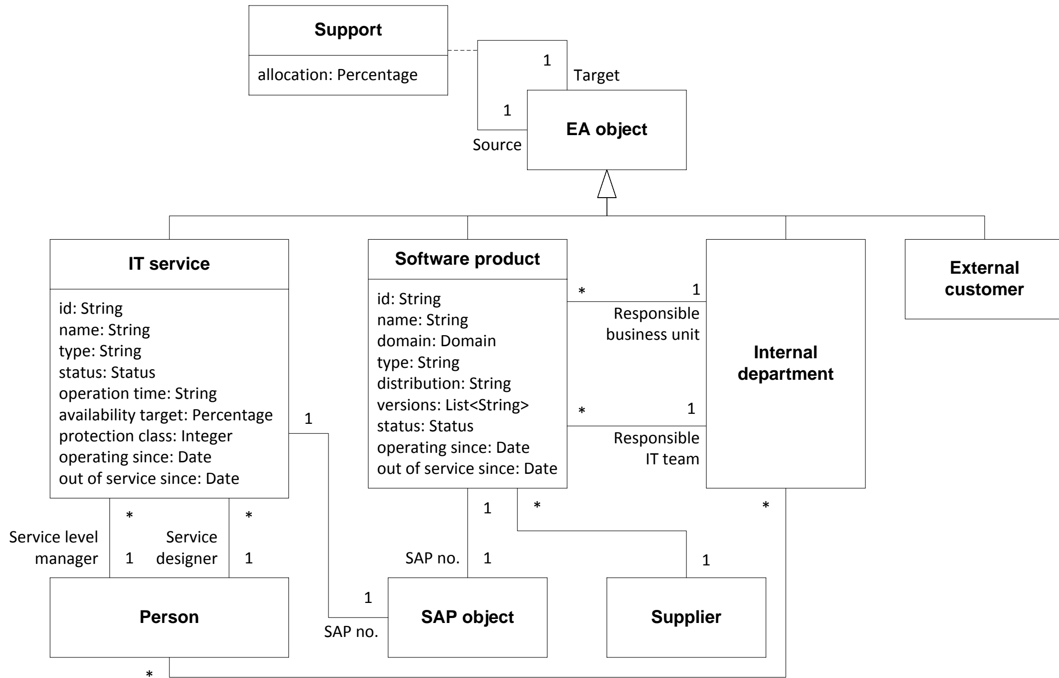


Figure 4.15.: Meta model

Offering the additional choice of a base version could confuse users.
Solution: As visualisations in ModelGlue are easily configurable, users might be given the possibility to choose whether to omit or to visualise base versions. User-specific preferences could be stored centrally in Tricia. Once the framework for such configuration capabilities is implemented, it would allow further individual adjustments.

Long-living/ignored conflicts ($\mathcal{F}_{B_6}$): Asked for the concept of ignoring conflicts as suggested by company A in $\mathcal{P}_{A_1}$, enterprise architect B initially was sceptical. He confirmed that conflicts may easily exist for several weeks, but most conflicts have to be solved eventually in order to assure the data quality within an EA model.

Conflict resolution table ($\mathcal{F}_{B_7}$): Even before being shown ModelGlue's conflict table (see figure 3.22), architect B suggested such a table with columns for the focal element, the attribute involved in a conflict and two columns showing the values from models A and B. For the actual resolution of model conflicts, enterprise architect B would favour this list.

4.2.4. Meta model and impacts on the technical complexity

Subsequent to interview I_{B_1}, company B provided us with en exemplary excerpt of their current EA model for further analysis. This data was used to draw conclusions about EA model complexity typical for a medium-sized organisation. Knowing model dimensions helps to outline realistic loads for EA model visualisations. Moreover, information about model complexity enables to adjust graphical means to the amount of data the visualisation

contains. This includes questions about whether to visualise data via graphs or via lists or tables as well as questions about how to reduce the complexity presented to the user, e.g. via a model element filter.

The meta model: The analysed data is part of an EAM scenario where costs of software products and IT services are allocated to further IT services. At the end of this cost allocation chain are internal departments or external customers that can be charged accurately. Heart of this chain is a meta model element called *'Support'*, which, as an association class[20], handles the allocation level between two 'EA objects'. This recursive association from a 'source' to a 'target' allows modelling transitive allocation scenarios.
Principal sources are *'software products'* or *'IT services'*. In the middle of an allocation chain also *'internal departments'* are modelled as cost sources. Intermediary targets can be *'IT services'* or *'internal departments'*. Final target is either an internal organisational unit or an *'external customer'*.

Technically speaking, cost allocation chains are actually graphs. The *'Support'* class specifies edges in the allocation graph, while *'EA objects'* form nodes. Cost calculation of single final 'target' nodes (only incoming and no outgoing edges) can be seen as trees with this final 'target' as root node. Similarly, only examining nodes originating from a single starting 'source' node (only outgoing and no incoming edges) also forms a tree.

Cost allocation graph example Application of the meta model described above can best be explained in an example. Figure 4.16 shows an exemplary excerpt a cost allocation graph. Because of the specified allocation level of 100% (see object 's1'), all costs of the software product 'TriciaSW' are transferred to the IT service 'TriciaITS'. Costs of the 'TriciaITS', in turn, are apportioned between 'Procurement' department (1,12%), 'Finance' department (3,3%), 'Controlling' department (0,63%) and other organisational units (omitted in the schematics) thus that they add up to 100%. In order to indicate that 'support' relationships are non-injective functions which, globally seen, form a graph, a further software service 'SAPservices' contributes to the costs of the 'controlling' department.

Graph complexity. The remainder of this section deals with the quantity structure of the EA model and impacts onto its visual representation. Like in the section evaluating technical complexity in company A (section 4.1.4) we try to evaluate hypotheses about element quantities which might be relevant for ModelGlue's different visualisation layers (published in [RM14]). Roth estimated that in layer one (schema graph) at most 100 classes ($\mathcal{H}_1$), each with up to 10 attributes ($\mathcal{H}_2$), have to be visualised. Layer two (instance list) might contain as much as 10.000 instances of a single object definition ($\mathcal{H}_3$). Finally, layer three (instance neighbourhood graph) should not comprise more than 10 model elements ($\mathcal{H}_4$).
A short glance at the meta model depicted in figure 4.15 confirms hypotheses $\mathcal{H}_1$ and $\mathcal{H}_2$

[20]See the UML specification for an explanation of the construct of association classes: [Ob11b], p. 44ff

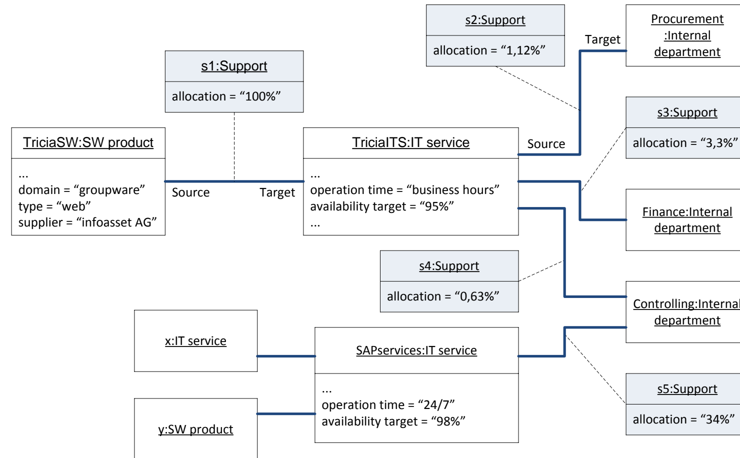


Figure 4.16.: Part of an exemplary cost allocation graph

as the EA model excerpt contains eight object definitions[21] with at most nine attribute definitions.

Graph complexity - a) 'Support': The object definition 'Support' forms the conceptual centre of the given EA model excerpt. In total, the model contains over 1000 instances conforming to the 'Support' schema. Each instance contains exactly two relationships to other instances (source and target). Hence, an instance neighbourhood graph (visualisation layer three) of a 'Support' element will always illustrate three model elements. These numbers are in line with Roth's hypotheses $\mathcal{H}_3$ and $\mathcal{H}_4$.
The sheer amount of over a thousand elements forbids any holistic graph visualisations. Consequently, ModelGlue presents the entirety of instances in a list view in layer two. But even this list view may exceed human comprehension for large numbers of elements. Figure 4.18 shows 1000 instances forced into the frame of a web browser window. These problems related to the visual complexity justify the efforts made to provide effective filtering capabilities (see section 3.8.4), especially for the instance list.

Graph complexity - b) 'Software product': The examined data source contains about 230 software products, which confirms $\mathcal{H}_3$. Every instance has a link to its 'responsible IT team' and 110 of them implement the relationship to a 'responsible business unit'. 150 of the 230 instances have a supplier and 70 are listed in SAP. Consequently, an average software product in company B has outgoing links to $\frac{230+110+150+70}{230} \approx 2,5$ model elements.
As on average a software product appears about three times as 'source' in 'Support' instances, the software product is aware of 3 incoming links. Summing up, including the 'Software product' instance itself, $2,5+3+1 \approx 6,5$ elements will have to be visualised in the instance neighbourhood graph. This is is well within the limits estimated by hypothesis $\mathcal{H}_4$. 'SAP objects' and especially 'Suppliers' could of course also be modelled as plain attribute values - then an instance neighbourhood graph would only comprise an average of 4,5 elements.

[21]...or nine if the underlying system would allow type hierarchies - which Tricia does not.

Figure 4.17.: Instance list (L2) with 200 instances

Figure 4.18.: Instance list (L2) with 1000 instances

Graph complexity - c) 'IT service': 200 IT services can be found in company B (consistent with $\mathcal{H}_3$). As those are compulsory relationships, every IT service has a link to its 'Service level manager', its 'Service designer' and its SAP reference. In contrast to software products, IT services are also listed as 'targets' in the 'Support' schema, hence they contain more incoming links. So on average, a total of 3 (outgoing links) + 3 (incoming links as 'source') + 4,5 (incoming links as 'target') + 1 (this instance) $\approx$ 11,5 elements will have to be visualised in the instance neighbourhood graph. This exceeds the ten elements estimated by hypothesis $\mathcal{H}_4$, but not significantly and therefore should not lead to any visual problems.

4.3. Comparison of the two cases

Following Runeson's principle of *comparative design* [RH09], this section tries to present similarities and differences in the feedback collected within the two case studies.

Table 4.3 compares prominent aspects evaluated in both companies. The first column provides the unique reference identifiers of each aspect.

Differences between the two cases have to be seen against the background of the different enterprise sizes (10.000 against 1.500 employees). Since company A maintains significantly more IT systems, their EA models are more complex than those of company B. As complex models often imply more model conflicts, ModelGlue's functionality can tap its full potential in large enterprises rather than in small ones.

Due to a more complex IT landscape, company A also employs more people who deal with EA management. The advanced model conflict task assign strategies shown in this thesis only make real sense if a large user base can be reached. Therefore, some of ModelGlue's concepts with regard to model conflict task dissemination and collaboration in general depend on a sufficiently large user base.

Apart from these complexity differences, ModelGlue's functionality was generally appreciated. Both cases showed that its functionality can support essential parts of enterprise architecture model management.

Table 4.3.: Comparison of prominent aspects of evaluation feedback received from companies A and B

Ref. Id	Aspect	Company A	Company B
$\mathcal{F}_{A_7}/\mathcal{F}_{B_3}$	Comprehensibility of the filter	Intuitive	Intuitive
$\mathcal{F}_{A_7}/\mathcal{F}_{B_3}$	Expressiveness of the filter	Complex models & complex queries require more	Sufficient
$\mathcal{F}_{A_7}/\mathcal{F}_{B_4}$	MxL filter	Power users like enterprise architects would use it	Too advanced, since there are too few power users
$\mathcal{F}_{A_0}/\mathcal{P}_{B_5}$	Base version	Valuable information	Too complex; 2-way is enough
$\mathcal{S}_{A_3}/\mathcal{S}_{B_5}$	Visualise constraint violations in the conflict dashboard	Is the responsibility of a different role; not EA	Would be handy to assess data and model quality
$\mathcal{F}_{A_5}/\mathcal{F}_{B_7}$	Conflict table vs. visualisation	Vis. for an overview and quality assurance, table for conflict resolution	Vis. for managers (overview); Conflict resolution in the table (by model experts)

4.4. Federated EA model management survey

After the case studies reported in the previous sections had been concluded, we conducted an online survey among EA experts on the topic of federated EA model management. As described in chapter 2, our goal was to validate...

- *a*) general aspects of federated EA model management,
- *b*) aspects of ModelGlue in particular, and finally
- *c*) findings derived from the conducted case studies.

By the time of the deadline of this thesis, 48 enterprise architecture experts had participated in this survey, 35 of which provided partial responses.

This section offers a brief evaluation of survey results with respect to technical aspects of federated EA model elements, since those were the focus of this thesis. Evaluation of governance aspects asked for in the survey can be found in [Kh14], a more detailed holistic evaluation in [Ro14]. The entire questionnaire of the online survey is printed in part B of the appendix. Due to the moderate response rate, most aspects presented in this section are evaluated qualitatively rather than quantitatively.

Dimensions and technical complexity of EA models: Asked for the number of federated information sources relevant for their EA model, respondents gave very deviating answers, ranging from a handful to three-digit numbers, revealing huge differences in EA model complexity between different enterprises (e.g. smaller and larger firms).

Model update frequency: Important for assessing the necessity of providing tool support and automation capabilities is the question about how often EA practitioners typically update their EA model by importing up-to-date information. 28 of 39 respondents stated that for technical information they would do so at least quarterly. Hence automation efforts are relevant in practice, especially for technical information.

Change suggestions: Conflicts may also be caused by relationships between objects. So the visualisation should be adapted to show conflicts associated with relationships. Currently, conflicts attached to relations can be detected because relations are implemented as $LinkValues$ in Tricia. However, those conflicts can only be illustrated like attributes. This is not seen as intuitive by the respondents of the survey.

A further general demand was to show more information about links as they play an important part in EA models.

4.5. Performance considerations

This chapter intends to provide insights into prominent performance characteristics of the ModelGlue implementation. It presents two major performance problems we initially experienced as well as solutions to these problems. General considerations about performance behaviour can be found in Roth's thesis [Ro14].

4.5.1. Base version extraction

Initially, ModelGlue suffered from slow model merging and differencing routines. After thorough investigation, the primary problem could be identified as the function that extracts the base version of model elements. As described in the design chapter of this thesis, both merge and differences algorithm heavily rely on having the base version of every model element at their disposal, since this base version allows the differentiation of change operation (e.g. delete/new) and is shown as additional information in visualisations.

Reason for this slow extraction of base versions is that Tricia is not designed to be a versioning system. Tricia allows the retraction of versions, but since in typical 'wiki' operation this scenario is employed infrequently and hence not critical, the system is not optimised in this regard.

Sequence diagram 4.19 shows a glimpse of the steps necessary for the extraction of a base version. Note that various levels of the Java stack have been omitted as they do not aid the general understanding of the process. This process presumes that the baseline time t_{base} is already known.

Triggered by the merge algorithm, the *ChangeSet* class is responsible for the base version extraction. Before the actual calculation begins, all past changesets of the respective model element have to be obtained. A so-called 'TenantResourcesManager' internally answers these queries. It first consults a $CacheManager$, whether changesets for the respective model elements exist in Tricia's global element cache. If not, changesets have to be retrieved from the database. The list of changesets of a model element returned by $TenantResources$ is sorted according to changeset timestamps (*ChangeSet.when*; see figure 3.11). Finally, on a temporary version of the model element, all changesets can be retracted until the latest changeset before t_b is found. This version is returned as 'base version'.

In a test scenario with one *Object* comprising four *ChangeSets*, the entire process of base version extraction took an average of 9,76 milliseconds[22]. In a merge process with a data load typically found in company A (3.000 model elements per information source; see section 4.1.4), this time aggregates to a tremendous length. Even if every model element only has two changesets, due to the worst case algorithmic complexity of $\mathcal{O}(n^2)$ in the merge algorithm, base version extraction alone would take about four days ($9,76ms * (3000 * 2)^2$).

Solution: As a solution for this problem, we changed ModelGlue's design with the goal that the time-consuming process of base version extraction should only be conducted once.

[22] All performance tests were executed on a laptop with a 2,6 GHz quad core processor and 8 GB RAM. Under realistic circumstances, i.e. on server hardware, response times could of course be reduced significantly.

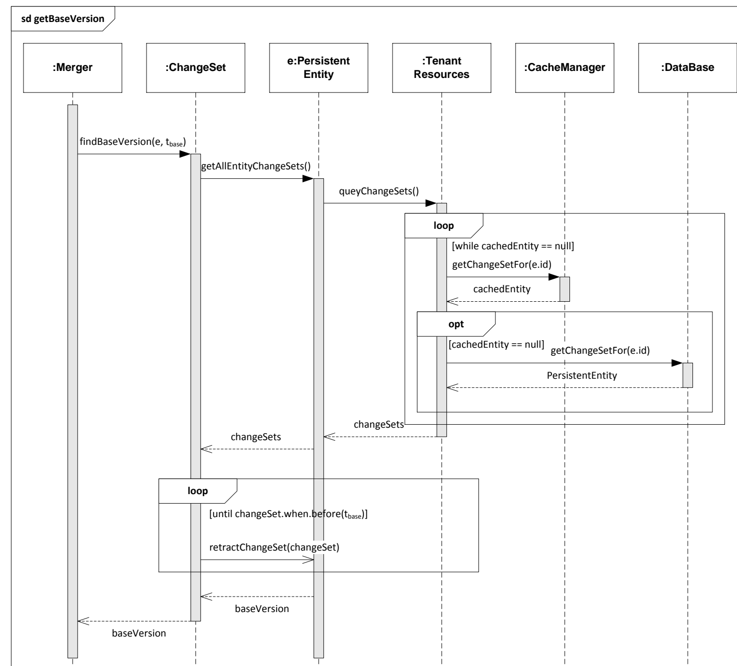


Figure 4.19.: Sequence diagram of extracting the base version of a model element

This could be achieved by materialising the base version after its first extraction. Every *ConflictListEntry* (see figure 3.6) now saves a *String* representation of its base version, if the base version can be represented as string (which is the case for most attributes).

This solution accelerates base version extraction by the factor 49, since the average retrieval of a single *String* value takes less than 0,2 milliseconds (compared to 9,76 milliseconds before), assuming that the base version has already been calculated. In a merge scenario as described above, the new implementation leads to slightly more than two hours ($0,2ms * (3000*2)^2 + 9,76ms * (3000*2)$), when all base versions have to be extracted exactly once.

As every *ConflictListEntry* is used at least a second time in the merge algorithm and several times more when it is visualised in the user interface, the materialisation of base values proved to be a significant improvement. Due to this obligatory usage of the serialised values, concerns about the antipattern 'overserialisation' (see e.g. Ford et al. [Fo12], p. 114ff) could be dispelled.

4.5.2. Visual element size calculation

Due to the huge amount of visual elements in complex EA model visualisations, calculation of model graphics initially was perceived as rather slow. Profiling of visualisation algorithms revealed the *Text* class as root of the problem. Figure 4.20 shows an exemplary profiling result where the positioning of texts alone made up 42% of the time needed for calculating a merge dashboard visualisation. The highlighted row indicates that a total of 20% of the time were needed only for the function $getRealTextHeight()$, which is essential for element positioning and particularly for avoiding text overlaps.

Name	Time (ms)	
de.tum.sebis.visualization.viewpoint.differences.**Window.visit**(AbstractVisualizationObjectVisitor)	3,485	87 %
de.tum.sebis.visualization.visualizationmodel.**CompositeVisualizationObject.visit**(AbstractVisualizationObjectVisitor)	3,485	87 %
de.tum.sebis.visualization.visualizationmodel.**CompositeVisualizationObject.visit**(AbstractVisualizationObjectVisitor)	3,485	87 %
de.tum.sebis.visualization.visualizationmodel.**CompositeSymbol.visit**(AbstractVisualizationObjectVisitor)	1,795	45 %
de.tum.sebis.visualization.visualizationmodel.interactive.interaction.**Click.visit**(AbstractVisualizationObjectVisitor)	1,690	42 %
de.tum.sebis.visualization.renderer.**AbstractVisualizationObjectVisitor.visit**(Click)	1,690	42 %
de.tum.sebis.visualization.visualizationmodel.interactive.interaction.**Show.visit**(AbstractVisualizationObjectVisitor)	1,690	42 %
de.tum.sebis.visualization.visualizationmodel.**CompositeVisualizationObject.visit**(AbstractVisualizationObjectVisitor)	1,690	42 %
de.tum.sebis.visualization.visualizationmodel.**CompositeVisualizationObject.visit**(AbstractVisualizationObjectVisitor)	1,690	42 %
de.tum.sebis.visualization.visualizationmodel.**CompositeVisualizationObject.visit**(AbstractVisualizationObjectVisitor)	1,690	42 %
de.tum.sebis.visualization.visualizationmodel.**Text.visit**(AbstractVisualizationObjectVisitor)	1,690	42 %
de.tum.sebis.visualization.renderer.**AbstractVisualizationObjectVisitor.visit**(Text)	894	22 %
de.tum.sebis.visualization.renderer.**SizeVisitor.visit**(Text)	795	20 %
de.tum.sebis.visualization.visualizationmodel.**Text.getRealTextHeight**()	795	20 %

Figure 4.20.: 42% of the time needed for a model visualisation being spent on placing text elements, 27% on calculating the text height

Text.getRealTextHeight() is a method which has to be called for every text fragment shown in a visualisation. It considers the comprised *String* value and further *Text* attributes, like size and weight of the font, and calculates the resulting height of the bounding box of the *Text*. This bounding box is vital for the layout of all visual elements.

Solution: After the identification of the fact that so much time is spent within a single method in the code, the apparent solution was to introduce caches to avoid unnecessary calculations. Since the same text fragments are often used multiple times (e.g. the text "Name:" almost certainly appears several times in every EA model visualisation), such a caching approach offers great potentials. Consequently, we implemented caches for all repetitive calculations on *Text* level which are used in the visual layout algorithms.

As a result, ModelGlue's time effort for calculating a visualisation with 3000 model elements could be cut by four. Figure 4.21 shows this reduction from initially 16 seconds to less than 4 seconds if caches are full. The curve measures several consecutive calculations of the same visualisation. Since e.g. after a system restart, initially all element caches are empty, the first visualisation may take up to 8 seconds. The knee in the curve after the third iteration indicates the point where all object caches are filled and a maximum number of elements can be retrieved from the cache. After this point, the number of cache hits per visualisation does not increase any further. This behaviour is a typical 'priming effect' systems show when first started ([Fo12], p. 172).

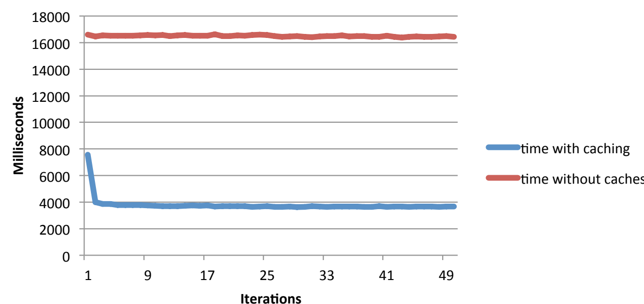


Figure 4.21.: Comparison of visualisation calculation times with and without caching

5. Related Work

Important steps of earlier work on topics related to federated EA model management are already described in the introduction (chapter 1). Besides the mentioned research stream, related work can be found in manifold disciplines on different levels of abstraction. Runeson and Höst distinguish between "a) earlier studies on the topic and b) theories on which the current study is based" [RH09].

A related case study was performed by Farwick et al. [Fa13b], who presents a textual EA modelling tool including architecture visualisations. As we know of no other case study having been conducted on federated EA model management before, the remainder of this section addresses aspect *b*), elemental theories this work is based on.

Apart from several explorative studies motivating automated EA documentation [Be12, HMR12, Ro13], notable publications include Buschle's work, who partially integrated system information relevant for the EA by extracting it from an SAP Enterprise Service Bus.

Important contributions on the field of automated enterprise architecture model maintenance were also reported by Farwick et al. ([Fa12] and [Fa14]).

A vital part of ModelGlue's functionality is the capability to merge EA models and thereby detect model conflicts.

Roots of the concepts employed for merging go back to Lippe [LVO92], who introduced an operation-based merge algorithm which detects conflicts. Several publications have extended and refined these concepts since then (see e.g. [CW98], [IN04], [Bl08] or [KHS09])

Recent research in the field of model merging, conflict detection and resolution within model-driven engineering can for instance be found in Altmanninger et al. [Al08] or Kelter et al. [KKK13]. Taentzer et al. [Ta10] perform both a state-based and an operation-based conflict detection approach in order to detect model conflicts. Wieland et al. [Wi13] were among the first who investigated collaborative conflict resolution in this regard.

Scientists at the University Augsburg are currently addressing federated EAM using ontologies and Semantic Web technologies with an approach called Semantic Enterprise Architecture Management (SEAM) [Ch13, DB14]. In this endeavour, Chen et al. distinguish between syntactic, structural and semantic model conflicts [Ch13].

6. Summary

This thesis consists of two basic parts. A detailed account of ModelGlue's functionality and the evaluation conducted in the form of two case studies in the industry. In addition to iterative interviews, much effort was spent on the assessment of technical complexity ModelGlue typically faces. In this regard also two initial performance issues are desribed.

6.1. Conclusions

In the two case studies conducted for this work we received mostly positive feedback on ModelGlue's concepts. The participating enterprise architects appreciated all core functionality we provide. Change suggestions on conceptual and implementation details were a valuable input for those improvements already realised and still are of great value for further enhancements and extensions. Further key findings include the following:

Model differences visualisations and the conflict management dashboard are of great value for data quality assurance of EA models.

The presented filter solutions provide intuitive means for the effective reduction of EA model complexity.

Many of ModelGlue's concepts depend on a certain organisational size in order to tap their full potential. Especially advanced task assignment scenarios require a sufficiently large user base.

Some of the implemented concepts are ahead of their time. Especially the collaborative conflict resolution support (see section 3.9) was appreciated by all interview partners, yet deemed to remain a vision for the next five to ten years.

6.2. Limitations

Due to limitations of the underlying platform, currently not all change operations can be detected. 'Move attribute' operations, for instance, would require far deeper adjustments in Tricia.

The current implementation only considers the immediate neighbourhood of model elements when trying to detect semantic impacts of model changes. Advanced scenarios could require more sophisticated conflict detection approaches examining also transitive relationships.

Learning strategies for automated conflict resolution implemented in ModelGlue are still very rudimentary as they are based only on a simple threshold with regard to one of the following three aspects: model, last editor, change time.

ModelGlue currently only contains rudimentary EA data import facilities. In order to serve in praxis, this aspect would have to be refined.

Since ModelGlue uses external graph layouting libraries, positioning of model elements in graphs is arbitrary and may vary each time the visualisation is rendered. This is perceived as not very intuitive since in most EA scenarios visual elements should be allocated according to a clear semantics (infrastructural elements at the bottom, then applications, finally business level at the top).

6.3. Outlook

As enterprise architecture management comprises many fields of activity, many further aspects could be adapted in ModelGlue.

Within the use cases currently addressed by ModelGlue, especially automated conflict resolution strategies offer significant potentials. Since the number of model conflicts triggered by EA model operations may reach considerably large values, additional advanced learning strategies could help to contain manual effort necessary for conflict resolution. Such strategies could be adopted from related research disciplines, particularly from database sciences.

Even artificial intelligence approaches could be applied for automated conflict pattern or conflict resolution pattern identification.

As different EAM scenarios generally require different merge strategies, domain-specific merge strategies could be implemented in addition to the strict and tolerant merge strategies presented in this thesis.

Bibliography

[Ac13] Achenbach, P.: *Framework for the strategic planning of enterprise architectures.* Master's thesis. Technische Universität München. 2013.

[AK03] Atkinson, C.; Kuhne, T.: *Model-driven development: a metamodeling foundation. Software, IEEE.* 20(5):36–41. 2003.

[Al08] Altmanninger, K.; Kappel, G.; Kusel, A.; Retschitzegger, W.; Seidl, M.; Schwinger, W.; Wimmer, M.: *AMOR–towards adaptable model versioning.* In *1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS.* volume 8. pages 4–50. 2008.

[Be12] Berneaud, M.; Buckl, S.; Fuentes, A.; Matthes, F.; Monahov, I.; Nowobilska, A.; Roth, S. et al.: *Trends for Enterprise Architecture Management and Tools.* Technical report. 2012.

[BGM87] Benbasat, I.; Goldstein, D. K.; Mead, M.: *The case research strategy in studies of information systems. MIS quarterly.* 11(3). 1987.

[Bl08] Blanc, X.; Mounier, I.; Mougenot, A.; Mens, T.: *Detecting model inconsistency through operation-based model construction.* In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on.* pages 511–520. IEEE. 2008.

[Bo90] Babbie, E. R.; others: *Survey research methods.* 1990.

[BSH86] Basili, V.; Selby, R.; Hutchens, D.: *Experimentation in software engineering. Software Engineering, IEEE Transactions on.* SE-12(7):733–743. 1986.

[Bü07] Büchner, T.: *Introspektive modellgetriebene Softwareentwicklung.* PhD thesis. Technische Universität München. 2007.

[Bu12] Buschle, M.; Ekstedt, M.; Grunow, S.; Hauder, M.; Matthes, F.; Roth, S.: *Automated Enterprise Architecture Documentation using an Enterprise Service Bus.* In *Proceedings of in Americas conference on Information Systems (AMCIS).* 2012.

[Ch13] Chen, W.; Hess, C.; Langermeier, M.; Stuelpnagel, J.; Diefenthaler, P.: *Semantic Enterprise Architecture Management.* In *Proceedings of the 15th International Conference on Enterprise Information Systems (ICEIS).* 2013.

[Co01] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C.; others: *Introduction to algorithms.* volume 2. MIT press Cambridge. 2001.

[CW98] Conradi, R.; Westfechtel, B.: *Version models for software configuration management. ACM Computing Surveys (CSUR).* 30(2):232–282. 1998.

[DB14] Diefenthaler, P.; Bauer, B.: *Gap Analysis in Enterprise Architecture using Semantic Web Technologies.* In *Multikonferenz Wirtschaftsinformatik (MKWI) 2014.* Universität Paderborn. 2014.

[DM78] De Marco, T.: *Structured analysis and systems specification. Yourdon Inc.* 1978.

[Fa12] Farwick, M.; Pasquazzo, W.; Breu, R.; Schweda, C. M.; Voges, K.; Hanschke, I.: *A Meta-Model for Automated Enterprise Architecture Model Maintenance.* In *EDOC.* pages 1–10. 2012.

[Fa13a] Farwick, M.; Breu, R.; Hauder, M.; Roth, S.; Matthes, F.: *Enterprise Architecture Documentation: Empirical Analysis of Information Sources for Automation.* In *46th Hawaii International Conference on System Sciences (HICSS), Maui, Hawaii.* 2013.

[Fa13b] Farwick, M.; Trojer, T.; Breu, M.; Ginther, S.; Kleinlercher, J.; Doblander, A.: *A Case Study on Textual Enterprise Architecture Modeling.* In *Proceedings of the 8th Workshop on Trends in Enterprise Architecture Research (TEAR).* 2013.

[Fa14] Farwick, M.; Schweda, C.; Breu, R.; Hanschke, I.: *A Situational Method for Semi-automated Enterprise Architecture Documentation. Software & Systems Modeling Systems Modeling (SOSYM).* 2014.

[FAW07] Fischer, R.; Aier, S.; Winter, R.: *A Federated Approach to Enterprise Architecture Model Maintenance.* In *Enterprise Modelling and Information Systems Architectures – Concepts and Applications, Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2007), St. Goar, Germany, October 8-9, 2007.* pages 9–22. St. Goar, Germany. 2007.

[FC93] Flood, R. L.; Carson, E. R.: *Dealing with complexity.* Springer. 1993.

[Fi00] Fielding, R. T.: *Architectural styles and the design of network-based software architectures.* PhD thesis. University of California. 2000.

[Fl01] Flyvbjerg, B.: *Making social science matter: Why social inquiry fails and how it can succeed again.* Cambridge university press. 2001.

[Fl06] Flyvbjerg, B.: *Five misunderstandings about case-study research. Qualitative inquiry.* 12(2):219–245. 2006.

[Fo12] Ford, C.; Gileadi, I.; Purba, S.; Moerman, M.: *Patterns for Performance and Operability: Building and Testing Enterprise Software.* CRC Press. 2012.

[Fr08] Friendly, M.: *Milestones in the history of thematic cartography, statistical graphics, and data visualization.* In *Seeing Science: Today.* American Association for the Advancement of Science. 2008.

[GMR12] Grunow, S.; Matthes, F.; Roth, S.: *Towards Automated Enterprise Architecture Documentation: Data Quality Aspects of SAP PI.* In *16th East European Conference on Advances in Databases and Information Systems (ADBIS).* pages 103–113. Pozna, Poland. 2012.

[Ha12] Hanschke, I.: *Enterprise Architecture Management - einfach und effektiv: Ein praktischer Leitfaden für die Einführung von EAM.* Carl Hanser Verlag GmbH & Company KG. 2012.

[Ha13a] Hanschke, I.: *Strategisches Management der IT-Landschaft: Ein praktischer Leitfaden für das Enterprise Architecture Management.* Carl Hanser Verlag GmbH & Company KG. 2013.

[Ha13b] Hauder, M.; Roth, S.; Pigat, S.; Matthes, F.: *Tool Support for Conflict Resolution of Models for Automated Enterprise Architecture Documentation.* In *ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013).* Miami, FL, USA. 2013.

[HMR12] Hauder, M.; Matthes, F.; Roth, S.: *Challenges for Automated Enterprise Architecture Documentation.* In *Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation.* pages 21–39. Springer. 2012.

[IN04] Ignat, C.-L.; Norrie, M. C.: *Operation-based versus state-based merging in asynchronous graphical collaborative editing.* In *Proc. 6th International Workshop on Collaborative Editing Systems, Chicago.* 2004.

[in14] infoAsset: *Tricia PIM.* `http://infoasset.de/loesungen.html`, Last checked: 11.4.2014 21:35. 2014.

[it14] iteratec: *iteraplan.* `https://www.iteraplan.de`, Last checked: 6.5.2014 14:00. 2014.

[Ja86] Jackson, P.: *Introduction to Expert Systems.* International Computer Science Series. Addison-Wesley. 1986. 9780201142235.

[jg14] jgraph: *mxGraph.* `http://www.jgraph.com/mxgraph.html`, Last checked: 22.04.2014 13:35. 2014.

[KAV05] Kaisler, S.; Armour, F.; Valivullah, M.: *Enterprise Architecting: Critical Problems.* Jan 2005.

[KE09] Kemper, A.; Eickler, A.: *Datenbanksysteme: Eine Einführung.* Oldenbourg Wissenschaftsverlag. 7th edition. 2009.

[Kh14] Khosroshahi, P. A.: *Evaluation of Governance- and Process Structures of the federated Enterprise Architecture Model Management.* Master's thesis. Technische Universität München (to appear). 2014.

[KHS09] Koegel, M.; Helming, J.; Seyboth, S.: *Operation-based conflict detection and resolution.* In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models.* pages 43–48. IEEE Computer Society. 2009.

[Ki02] Kitchenham, B.; Pfleeger, S.; Pickard, L.; Jones, P.; Hoaglin, D.; El Emam, K.; Rosenberg, J.: *Preliminary guidelines for empirical research in software engineering. Software Engineering, IEEE Transactions on.* 28(8):721–734. 2002.

[Ki12] Kirschner, B.: *Graphical Interaction on Enterprise Architecture Visualisations.* Bachelor's thesis. Technische Universität München. 2012.

[KKK13] Kelter, U.; Kehrer, T.; Koch, D.: *Patchen von Modellen. Proc. Software Engineering.* 26(01.03):2013. 2013.

[Ko10] Koegel, M.; Herrmannsdoerfer, M.; Li, Y.; Helming, J.; David, J.: *Comparing State- and Operation-Based Change Tracking on Models.* In *14th IEEE International Enterprise Distributed Object Computing (EDOC) Conference.* pages 163–172. Los Alamitos, CA, USA. 2010. IEEE Computer Society.

[KPP95] Kitchenham, B.; Pickard, L.; Pfleeger, S. L.: *Case studies for method and tool evaluation. Software, IEEE.* 12(4):52–62. 1995.

[KR14] Kirschner, B.; Roth, S.: *Federated Enterprise Architecture Model Management: Collaborative Model Merging for Repositories with Loosely Coupled Schema and Data.* In *Multikonferenz Wirtschaftsinformatik (MKWI) 2014.* pages 2164–2174. Universität Paderborn. 2014.

[La05] Lankhorst, M.: *Enterprise Architecture at Work.* Springer-Verlag. 2005.

[Le89] Lee, A. S.: *A scientific methodology for MIS case studies. MIS quarterly.* pages 33–50. 1989.

[LSS05] Lethbridge, T. C.; Sim, S. E.; Singer, J.: *Studying software engineers: Data collection techniques for software field studies. Empirical software engineering.* 10(3):311–341. 2005.

[LVO92] Lippe, E.; Van Oosterom, N.: *Operation-based merging.* In *ACM SIGSOFT Software Engineering Notes.* volume 17. pages 78–87. ACM. 1992.

[Ma08a] Matthes, F.: *Informatik-Spektrum.* volume 31. chapter Softwarekartographie, pages 527–536. Springer-Verlag. 2008.

[Ma08b] Matthes, F.; Buckl, S.; Leitel, J.; Schweda, C. M.: *Enterprise Architecture Management Tool Survey 2008.* Technical report. Technische Universität München. 2008.

[Me02] Mens, T.: *A state-of-the-art survey on software merging. Software Engineering, IEEE Transactions on.* 28(5):449–462. 2002.

[MNS11] Matthes, F.; Neubert, C.; Steinhoff, A.: *Hybrid Wikis: Empowering Users to Collaboratively Structure Information.* In *Proceedings of the 6th International Conference on Software and Data Technologies (ICSOFT).* 2011.

[Mo97] Montgomery, D. C.: *Design and analysis of experiments.* volume 7. Wiley New York. 1997.

[Mo09] Moody, D.: *The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. Software Engineering, IEEE Transactions on.* 35(6):756–779. Nov 2009.

[MRM13] Monahov, I.; Reschenhofer, T.; Matthes, F.: *Design and prototypical implementation of a language empowering business users to define Key Performance Indicators for Enterprise Architecture Management.* In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International.* pages 337–346. IEEE. 2013.

[My75] Myers, G.: *Reliable software through composite design.* Van Nostrand Reinhold. 1975. 9780442256203.

[Ne12] Neubert, C.: *Facilitating Emergent and Adaptive Information Structures in Enterprise 2.0 Platforms.* PhD thesis. Universitätsbibliothek der TU München. 2012.

[Ob11a] Object Management Group: *Business Process Model and Notation (BPMN), Version 2.0.* Technical report. http://www.omg.org/spec/BPMN/2.0/. 2011.

[Ob11b] Object Management Group: *OMG Unified Modeling Language (OMG UML), Superstructure.* Technical report. http://www.omg.org/spec/UML/2.4.1/. 2011.

[Ob14] Object Management Group: *Meta Object Facility (MOF).* Technical report. http://www.omg.org/spec/MOF/. 2014.

[Re13] Reschenhofer, T.: *Design and prototypical implementation of a model-based structure for the definition and calculation of Enterprise Architecture Key Performance Indicators.* Master's thesis. Technische Universität München. 2013.

[RH09] Runeson, P.; Höst, M.: *Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering.* 14(2):131–164. 2009.

[RHM13a] Roth, S.; Hauder, M.; Matthes, F.: *Collaborative Evolution of Enterprise Architecture Models.* In *8th International Workshop on Models at Runtime (Models@run.time).* Miami, USA. 2013.

[RHM13b] Roth, S.; Hauder, M.; Matthes, F.: *Facilitating Conflict Resolution of Models for Automated Enterprise Architecture Documentation.* In *Nineteenth Americas Conference on Information Systems (AMCIS).* 2013.

[RM14] Roth, S.; Matthes, F.: *Visualizing Differences of Enterprise Architecture Models.* In *International Workshop on Comparison and Versioning of Software Models (CVSM) at Software Engineering (SE).* Kiel, Germany. 2014.

[Ro02] Robson, C.: *Real world research.* 2nd Edition. Blackwell Publishing. Malden. 2002.

[Ro13] Roth, S.; Hauder, M.; Farwick, M.; Breu, R.; Matthes, F.: *Enterprise Architecture Documentation: Current Practices and Future Directions.* In *11th International Conference on Wirtschaftsinformatik (WI), Leipzig, Germany.* 2013.

[Ro14] Roth, S.: *Federated Enterprise Architecture Model Management — Conceptual Foundations, Collaborative Model Integration, and Software Support.* PhD thesis. Technische Universität München (to appear). 2014.

[Sc06] Schweda, C. M.: *Architektur eines Visualisierungswerkzeugs für Anwendungslandschaften - Anforderung und Realisierung von Kernkompetenzen.* Master's thesis. Technische Universität München. 2006.

[Sc13] Schrade, T.: *A Visual Tool for Conflict Resolution in EA Repositories.* Bachelor's thesis. Technische Universität München. 2013.

[SCT01] Shull, F.; Carver, J.; Travassos, G. H.: *An Empirical Methodology for Introducing Software Processes. SIGSOFT Softw. Eng. Notes.* 26(5):288–296. Sep 2001.

[SDJ07] Sjoberg, D. I.; Dyba, T.; Jorgensen, M.: *The future of empirical methods in software engineering research.* In *Future of Software Engineering, 2007. FOSE'07.* pages 358–378. IEEE. 2007.

[Se99] Seaman, C. B.: *Qualitative methods in empirical studies of software engineering. Software Engineering, IEEE Transactions on.* 25(4):557–572. 1999.

[SI11] Steele, J.; Iliinsky, N.: *Designing Data Visualizations: Representing Informational Relationships.* O'Reilly Media, Inc. 2011.

[SMR12] Schaub, M.; Matthes, F.; Roth, S.: *Towards a Conceptual Framework for Interactive Enterprise Architecture Management Visualizations.* In *Modellierung.* pages 75–90. 2012.

[St95] Stake, R.: *The Art of Case Study Research.* SAGE Publications. 1995.

[Ta10] Taentzer, G.; Ermel, C.; Langer, P.; Wimmer, M.: *Conflict detection for model versioning based on graph modifications.* In *Graph Transformations.* pages 171–186. Springer. 2010.

[TSO11a] *ITIL Service Design: 2011.* Number 2 in Best Management Practice. TSO, The Stationery Office. 2011.

[TSO11b] *ITIL Service Strategy 2011.* Number 1 in Best Management Practice. TSO, The Stationery Office. 2011.

[VSB99] Van Solingen, R.; Berghout, E.: *The Goal/Question/Metric Method: a practical guide for quality improvement of software development.* volume 2. McGraw-Hill London. 1999.

[WHH03] Wohlin, C.; Höst, M.; Henningsson, K.: *Empirical Research Methods in Software Engineering.* In (Conradi, R.; Wang, A., Ed.): *Empirical Methods and Studies in Software Engineering.* volume 2765 of *Lecture Notes in Computer Science.* pages 7–23. Springer Berlin Heidelberg. 2003.

[Wi07] Wittenburg, A.: *Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften.* PhD thesis. TU München. 2007.

[Wi10] Winter, K.; Buckl, S.; Matthes, F.; Schweda, C. M.: *Investigating the State-of-the-Art in Enterprise Architecture Management Methods in literature and Practice. MCIS*. 90. 2010.

[Wi13] Wieland, K.; Langer, P.; Seidl, M.; Wimmer, M.; Kappel, G.: *Turning conflicts into collaboration. Computer Supported Cooperative Work (CSCW)*. 22(2-3):181–240. 2013.

[WR09] Weill, P.; Ross, J.: *IT Savvy: What Top Executives Must Know to Go from Pain to Gain*. Harvard Business Press. 2009.

[Yi03] Yin, R.: *Case Study Research: Design and Methods*. 3rd Edition. SAGE Publications. 2003.

A. Conflict classification matrices

Figure A.1 shows the entire pre-defined strict conflict classification matrix. Figure A.2 shows the entire pre-defined tolerant conflict classification matrix.

Tables A.1, A.2 and A.3 briefly explain the reasoning behind the data/data part of the pre-defined strict conflict classification matrix we provide. Reasoning for the rest of the matrix follows similar patterns. The last column marks cases which are currently not detectable in ModelGlue on Tricia. See section 3.5.1 for reasons.

	object update	object delete	object create	object move	object use	attribute update	attribute delete	attribute create	attribute move	attribute use	value update	value delete	value create	value move	value use
object update	conflict														
object delete	approve	none													
object create	conflict	none	conflict												
object move	validate	approve	conflict	conflict											
object use	validate	approve	none	validate	none										
attribute update	none	approve	none	validate	validate	conflict									
attribute delete	none	none	none	none	validate	approve	none								
attribute create	none	approve	none	validate	validate	conflict	none	conflict							
attribute move	none	none	none	none	none	validate	validate	conflict	conflict						
attribute use	none	none	none	none	none	validate	validate	none	none	none					
value update	validate	approve	none	validate	validate	none	approve	none	none	none	conflict				
value delete	none	none	none	none	validate	validate	none	none	none	none	none	none			
value create	none	approve	none	validate	validate	validate	none	none	none	none	none	none	none		
value move	none	none	none	none	none	none	none	none	none	none	none	none	none	none	
value use	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none
objectdefinition update	validate	none	validate	none	none	validate	none	validate	none	none	validate	none	validate	none	none
objectdefinition delete	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none
objectdefinition create	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none
objectdefinition move	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none
objectdefinition use	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none
attributedefinition update	none	none	validate	none	none	none	none	none	none	none	validate	none	validate	none	none
attributedefinition delete	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none
attributedefinition create	none	none	none	none	none	conflict	none	conflict	none	none	none	none	none	none	none
attributedefinition move	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none
attributedefinition use	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none

	objectdefinition update	objectdefinition delete	objectdefinition create	objectdefinition move	objectdefinition use	attributedefinition update	attributedefinition delete	attributedefinition create	attributedefinition move	attributedefinition use
objectdefinition update	conflict									
objectdefinition delete	approve	none								
objectdefinition create	none	none	conflict							
objectdefinition move	none	none	none	none						
objectdefinition use	validate	approve	none	validate	none					
attributedefinition update	none	none	none	none	none	conflict				
attributedefinition delete	none	none	none	none	none	approve	none			
attributedefinition create	validate	none	none	none	none	conflict	none	conflict		
attributedefinition move	none	none	none	none	none	none	none	none	none	
attributedefinition use	none	none	none	none	none	validate	approve	none	validate	none

Figure A.1.: Pre-defined strict conflict classification matrix

	object update	object delete	object create	object move	object use	attribute update	attribute delete	attribute create	attribute move	attribute use	value update	value delete	value create	value move	value use	objectdefinition update	objectdefinition delete	objectdefinition create	objectdefinition move	objectdefinition use	attributedefinition update	attributedefinition delete	attributedefinition create	attributedefinition move	attributedefinition use
object update	conflict																								
object delete	approve	none																							
object create	conflict	none	conflict																						
object move	none	approve	conflict	conflict																					
object use	none	none	none	none	none																				
attribute update	none	approve	none	none	none	conflict																			
attribute delete	none	none	none	none	none	approve	none																		
attribute create	none	approve	none	none	none	conflict	none	conflict																	
attribute move	none	none	none	none	none	none	none	conflict	conflict																
attribute use	none	none	none	none	none	none	none	none	none	none															
value update	none	approve	none	none	none	none	approve	none	none	none	conflict														
value delete	none	none	none	none	none	none	none	none	none	none	none	none													
value create	none	approve	none	none	none	none	none	none	none	none	none	none	none												
value move	none	none	none	none	none	none	none	none	none	none	none	none	none	none											
value use	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none										
objectdefinition update	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	conflict									
objectdefinition delete	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	approve	none								
objectdefinition create	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	conflict							
objectdefinition move	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none						
objectdefinition use	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	approve	none	none	none					
attributedefinition update	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	conflict				
attributedefinition delete	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	approve	none			
attributedefinition create	none	none	none	none	none	conflict	none	conflict	none	none	none	none	none	none	none	none	none	none	none	none	conflict	none	conflict		
attributedefinition move	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	
attributedefinition use	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	none	approve	none	none	none

Figure A.2.: Pre-defined tolerant conflict classification matrix

Table A.1.: Reasoning behind the strict conflict classification matrix (part 1)

op_1/obj_1	op_2/obj_2	**Precondition**	**Task Type**	**Reason**	
Object/Object:					
Update Object	Update Object		Conflict	Take which one?	
Update Object	Delete Object		Approve	New info (update) could lead to different decision?	
Update Object	Create Object	$e_1.name \equiv e_2.name \wedge e_1.context \equiv e_2.dest$	Conflict	Name collision when created in the same context (e.g. same parent object)	
Update Object	Move Object		Validate	Is the update still valid in the new context?	
Update Object	Use Object		Validate	Does the updated e_1 still meet the demands to be used by e_2?	
Delete Object	Move Object		Approve	Maybe the object makes sense in the new context? So maybe keep it?	
Delete Object	Use Object		Approve	Does the user of op_2 need e_1 or is the deletion OK?	
Create Object	Create Object	$e_1.name \equiv e_2.name \wedge e_1.context \equiv e_2.context$	Conflict	Name collision when created in the same context (e.g. same parent object)	
Create Object	Move Object	$e_1.name \equiv e_2.name \wedge e_1.context \equiv e_2.dest$	Conflict	Name collision, when e_2 is moved to $e_1.context$	
Move Object	Move Object	$e_1.dest \not\equiv e_2.dest$	Conflict	Moved to different destinations. Keep both? Only one correct?	
Move Object	Use Object		Validate	Is the link to e_1 still OK considering its new context?	
Attribute/Attribute:					
Update Attribute	Update Attribute		Conflict	Take which on?	
Update Attribute	Delete Attribute		Approve	New info (update) could lead to different decision?	
Update Attribute	Create Attribute	$e_1.name \equiv e_2.name$	Conflict	Name collision	
Create Attribute	Create Attribute	$e_1.name \equiv e_2.name$	Conflict	Name collision	
Create Attribute	Move Attribute	$e_1.name \equiv e_2.name$	Conflict	Name collision (move not detectable in Tricia)	×
Move Attribute	Move Attribute	$e_1.name \equiv e_2.name$	Conflict	Name collision	×

Table A.2.: Reasoning behind the strict conflict classification matrix (part 2)

op_1/obj_1	op_2/obj_2	**Precondition**	**Task Type**	**Reason**	
Attribute/Attribute:					
Move Attribute	Use Attribute		Validate	see object move/use	×
Update Attribute	Use Attribute		Validate	see object move/use	×
Move Attribute	Delete Attribute		Validate	see object move/delete	×
Delete Attribute	Use Attribute		Validate	see object delete/use	×
Object/Attribute:					
Delete Object	Update Attribute		Approve	New info (update) could lead to different decision?	
Delete Object	Create Attribute		Approve	New info → still delete?	
Move Object	Update Attribute		Validate	Updated attribute still valid in new context?	
Move Object	Create Attribute		Validate	Updated attribute still valid in new context?	
Use Object	Update Attribute		Validate	Object still usable with altered attribute?	
Use Object	Delete Attribute		Validate	Object still usable without this attribute?	
Use Object	Create Attribute		Validate	Object still usable with this new attribute?	
Value/Value:					
Update Value	Update Value		Conflict	Conflicting values	
Attribute/Value:					
Update Value	Delete Attribute		Approve	Can the attribute be kept considering the updated information?	
Create Value	Update Attribute		Validate	Does the new value violate a constraint (e.g. ‘exactly one value’)?	
Delete Value	Update Attribute		Validate	Does the deletion violate a constraint (e.g. ‘min. one value’)? Delete Value is detectable if it is the last value	×

Table A.3.: Reasoning behind the strict conflict classification matrix (part 3)

op_1/obj_1	op_2/obj_2	**Precondition**	**Task Type**	**Reason**	
Object/Value:					
Update Value	Update Object		Validate	Does one change impact the other?	
Update Value	Delete Object		Approve	Can the object be kept considering the updated information?	
Create Value	Delete Object		Approve	Can the object be kept considering the new information?	
Move Object	Update Value		Validate	Updated value still valid in new context?	
Move Object	Create Value		Validate	Updated value still valid in new context?	
Use Object	Update Value		Validate	Object still usable with altered value?	
Use Object	Delete Value		Validate	Object still usable without this value? Delete Value is detectable if it is the last value	×
Use Object	Create Value		Validate	Object still usable with this new value?	

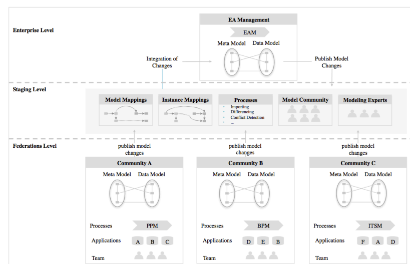
B. Questionnaire

The following pages show the questionnaire conducted for this work. High quality versions of images concerning ModelGlue and its implementation and visualisations can be found in the main part of this thesis. Images of the processes can be found in Aleatrati's thesis [Kh14].

Question Group #1: Scope of EA Models

We assume that an organization is divided into federated organizational units (called 'communities'). The communities operate autonomously, i.e. define their own processes, run specialized applications to fulfill their business needs and maintain own meta and data models. EA information about the specific communities can be retrieved from respective meta and data models of the communities. The federated EA model management approach aims at aggregating all EA relevant information within the meta and data models of the respective communities in one holistic and integrated EA model.

The illustration gives an overview of the approach.



Q1: Does your company perform federated EA model management or similar activities?
O yes
O no

Q2: How many different information sources contain data that is relevant for your EA?
Only numbers may be entered in this field:_______

Q3: How many different information sources are actually integrated into your holistic EA model?
Only numbers may be entered in this field:_______

Q4: How often is the EA model updated by importing up-to-date information...

	Weekly	Once a month	Quarterly	Once a year	Less frequently
...from infrastructure-level information sources?	O	O	O	O	O
...from application-level information sources?	O	O	O	O	O
...from business-level information sources?	O	O	O	O	O

Q5: How many schema elements on meta model level does your EA model comprise?
O n <= 5
O 5 < n <= 10
O 10 < n <= 25
O 25 < n <= 75
O n > 75

Q6: On average, how many instances exist in your company...

	n <= 25	25 <= n <= 100	100 < n <=1000	1000 <= n <= 10000	n > 10000
..on business level (e.g. 'Processes')?	O	O	O	O	O
...on IT level (e.g. 'Applications')?	O	O	O	O	O
..on infrastructure level (e.g. 'Servers')?	O	O	O	O	O

Q7: How can objects be mapped across application and division borders?
O Via unique foreign keys (automatically)
O Via exact name matching (automatically)
O Via exact name matching (manually)
O Via similarity matching (automatically)
O Via similarity matching (manually)
O Other:_______

Question Group #2: Alignment of Terminology

Before starting to thinking about designing a federated EA model management, an enterprise has to ensure a standardized terminology, e.g. for IT services, processes, platforms, applications, etc.

The figure below suggests a process to establish a standardized terminology of IT products within the organization.

Q1: Do you agree with the suggested process?

	Strongly Agree				Strongly Disagree
All necessary activities are included within the process.	O	O	O	O	O
We would perform a more pragmatic approach.	O	O	O	O	O

Q2: What would you improve within the process? ____

Q3: Within each activity, there might be disputes between communities or further reasons that could lead to a 'show stopper'. We advocate escalation paths to the EA Board within each activity.

We assume that the EA Board has upper Management support and is authorized to issue directives to the respective parties. Do you agree with this tough regulation (please justify your choice shortly in the comment field)?
O Yes
O No
____

Q4: How many months would you spend for the 'general setup activities' (Strategy / Scope Definition until Definition of a roadmap)?
Only numbers may be entered in this field:______

Q5: How many months would you spend to align the terminology for one meta model element (e.g. 'Application')?
Only numbers may be entered in this field:______

Q6: How many months would you spend for the complete alignment process?
Only numbers may be entered in this field:______

Question Group #3: Mapping of new instances

We assume that instances from the communities need to be added or updated into the EA model (e.g. a community implements a new application). Depending on the EA model change, further reconciliations and Data Quality (DQ) assessments are necessary.

The following figure gives an overview of mapping new instances into the federated EA model:

Q1: Which roles would you assign to each action?

	Modeling Expert	Business Architect	Domain Architect	Data Steward	Data Owner	Decision Maker	EA Coordinator	Enterprise Architect	Other	No answer
1	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O

Q2: Action 2 (Run DQ – Assessment) includes data quality assurance activities (missing granularity of data, wrong format of data, conflicts between instances, etc.) before uploading new instances into the EA Model.

We distinguish between conflicts on different levels. Firstly, conflicts may occur between meta model and data model elements (Meta model vs. Data).

Example: Community A created a new instance (data model level) of the type 'Mainframe', while community B deleted the schema element 'Mainframe'.

Have you ever been confronted with Meta model vs. Data conflicts?
O Yes
O No

Q3: How often does Meta model vs. Data conflicts occur?
O ≥ 5 times during one import process
O < 5 times during one import process
O Once every few import processes
O Never
O No answer

Q4: How many percent of Meta model vs. Data conflicts could possibly be solved automatically and how many have to be solved manually?
Only numbers may be entered in these fields.
Automatically: ______
Manually: ______

Q5: Another kind of conflicts that might occur during the mapping of new instances are conflicts between data model and data model elements (Data vs. Data).

Example: Community A changed the attribute 'Replacement cost' of application 'JBoss 7.1.1' to the value '10.000€', while community B estimated it to '12.000€'.

Have you ever been confronted with Data vs. Data conflicts?
O Yes
O No

Q6: How often do Data vs Data conflicts occur?
O ≥ 5 times during one import process
O < 5 times during one import process
O Once every few import processes
O Never
O No answer

Q7: How many percent of Data vs. Data conflicts could possibly be solved automatically and how many have to be solved manually?
Only numbers may be entered in these fields.
Automatically: ______
Manually: ______

Question Group #4: Change of Meta Model

We assume that a community changes his meta model and the enterprise architects want to implement the changes within the federated EA model. The following figure gives an overview of the performed process when changing the Meta Model (We assume that changes within the meta model of the EA model can only be performed by the EA Team):

Q1: Which roles would you assign to each action?

	Modeling Expert	Business Architect	Domain Architect	Data Steward	Data Owner	Decision Maker	EA Coordinator	Enterprise Architect	Other	No answer
1	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O

Q2: When uploading meta model changes from a community to the EA model, conflicts might occur. Conflicts between the meta models of two (or more) communities are referred to as 'Meta model vs. Meta model' conflicts.

Example: Community A changed a relationship between the schema (meta model) elements 'Application' and 'Platform', while community B deleted this relationship.

Have you ever been confronted with Meta model vs. Meta model conflicts?
O Yes
O No

Q3: How Often do Meta model vs. Meta model conflicts occur?
O Within every meta model change
O Once every few meta model changes
O Never
O No answer

Q4: How many percent of meta model conflicts could possibly be solved automatically and how many have to be solved manually? Only numbers may be entered in these fields.
Automatically: ______
Manually: ______

Question Group #5: Further characteristics

Q1: In terms of communication, we distinguish between three possibilities:
A) Only communities publish model changes to the EA model (unidirectional)
B) EA team communicates EA model changes to communities interpersonally (bidirectional)
C) EA team can manipulate information within the information systems of the communities manually (by person) or automatically (by machine) (bidirectional)

	A	B	C
What kind of communication would you advocate in terms of model changes?	O	O	O

Q2: We assume that the meta model of a community has changed and the new information will be transferred to the EA model. Some organizations prefer to view the differences between the current EA model and the EA model that includes the new data.

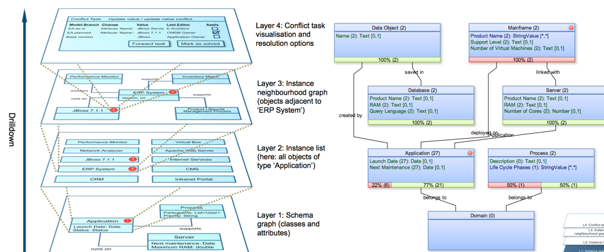
Would you perform this difference view automatically (in a tool) or manually (e.g. via Excel)? Please describe shortly who is in charge for this activity.
O Automatically
O Manually
O We do not conduct this activity (no comment needed)
____

Q3: Are you familiar with the concept of ontologies?
O Yes
O No

(When Q3 = Yes)
Q4: Would you make use of ontologies to ensure an automated EA documentation (Please explain your choice shortly within the comment field)?
O Yes
O No
____

Question Group #6: EA model visualization (layer 1)

Since conflicts may occur on different levels (schema or instance), we provide a set of visualizations depicting variable angles of abstraction. Starting with a schema graph, users can navigate through the four layers, retrieving information necessary for solving conflict tasks.



Layer 1 (schema graph): Each box in the following visualization depicts one schema element (type or class with its attributes) and a bar at the bottom, which indicates how many of the instances (data model level) of this type are involved in conflicts. Conflicts on schema level are visualized by red exclamation marks.

Q1: Do you understand the concept in this view?
O Yes
O No

Q2: This visualization…

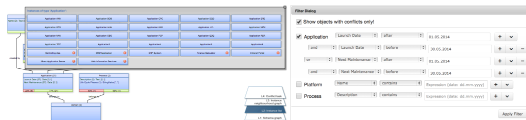
	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
… is intuitive.	O	O	O	O	O
...can cope with the complexity of an EA model.	O	O	O	O	O
...is helpful for the Enterprise Architect (technical perspective).	O	O	O	O	O
...is helpful for EA Stakeholders (business perspective).	O	O	O	O	O

Q3: What would you improve in this view? ____

Q4: Which concepts would you visualize? ____

Question Group #7: EA model visualization (layer 2)

Layer 2 (instance list): A click on one of the schema elements (types) opens an overview of all instances of this type. Red exclamation marks indicate conflicts within this instance.



Q1: Do you understand the concept in this view?
O Yes
O No

Q2: This visualization…

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
… is intuitive.	O	O	O	O	O
...can cope with the complexity of an EA model.	O	O	O	O	O
...is helpful for the Enterprise Architect (technical perspective).	O	O	O	O	O
...is helpful for EA Stakeholders (business perspective).	O	O	O	O	O

Q3: How many objectives would typically be shown in such an instance list?
O n <= 25
O 25 <= n <= 100
O 100 <= n <= 1000
O 1000 <= n <= 10000
O n >= 10000

Q4: Filter: As especially in the instance list visualization the amount of objects may become overwhelming, the following filter seeks to alleviate this problem. This example filters for instances of type 'Application' that will be launched or maintained in May.

This filter…

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
Is intuitive	O	O	O	O	O
fulfils your demands for filtering use cases.	O	O	O	O	O
...is helpful for a power user.	O	O	O	O	O
...is helpful for casual users.	O	O	O	O	O

Q5: What would you improve in the filter? ____

Q6: The following visualization shows an alternative implementation of the same filter query as above, using a model-based expression language (MxL). MxL allows very sophisticated queries.

```
find(Application)
  .where(("Launch Date" > "01.05.2014" and "Launch Date" < "30.05.2014") or
         ("Next Maintenance" > "01.05.2014" and "Next Maintenance" < "30.05.2014"))
```

Filtering via MxL...

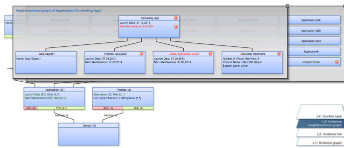
	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
Is intuitive	0	0	0	0	0
...is helpful for a power user.	0	0	0	0	0
...is helpful for casual users.	0	0	0	0	0

Q7: Would you filter rather via MxL than via the filter interface shown before?
0 Yes
0 No

Q8: What would you improve in the instance list visualization (layer 2)? ________________________________

Question Group #8: EA model visualization (layer 3)

Layer 3 (instance neighborhood graph): Another click on one instance (here the element "Controlling App") visualizes all linked instances to show contextual information of the conflicting element.



Q1: Do you understand the concept in this view?
0 Yes
0 No

Q2: Do you need such contextual information?
0 Yes
0 No

Q3: This visualization...

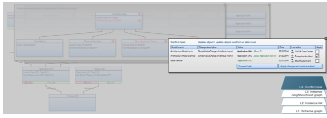
	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
... is intuitive.	0	0	0	0	0
...can cope with the complexity of an EA model.	0	0	0	0	0
Contextual information is relevant to resolve model conflicts.	0	0	0	0	0
...is helpful for the Enterprise Architect (technical perspective).	0	0	0	0	0
...is helpful for EA Stakeholders (business perspective).	0	0	0	0	0

Q4: How many objects would typically be shown in such a neighborhood graph?
0 n <= 5
0 5 <= n <= 10
0 10 <= n <= 25
0 25 <= n <= 75
0 n >= 75

Q5: What would you improve in this view? ____

Question Group #9: EA model visualization (layer 4)

Layer 4 (conflicts): Clicking on an exclamation mark (here of the instance "JBoss Application Server") shows information about this conflict. The three lines symbolize the two conflicting versions of the model plus a base version (the last version the two models had in common). Each line names the conflicting attribute, the change that led to the conflict, date of the last change, its user and the means to solve the conflict by applying one of the versions.



Q1: Do you understand the concept in this view?
O Yes
O No

Q2: This visualization…

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
… is intuitive.	O	O	O	O	O
...of changes via Text-compare colour coding is intuitive.	O	O	O	O	O
...is helpful for the Enterprise Architect (technical perspective).	O	O	O	O	O
...is helpful for EA Stakeholders (business perspective).	O	O	O	O	O

Q3: How important is he following meta-information about a conflict:

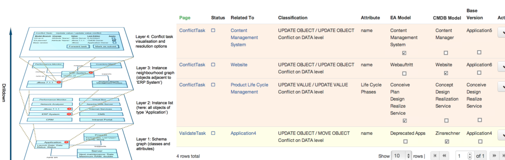
	Very important	Important	Neutral	Slightly important	Not important
Information source of the model	O	O	O	O	O
Date	O	O	O	O	O
Time	O	O	O	O	O
Last editor	O	O	O	O	O
Responsible role for this object	O	O	O	O	O
Previous value	O	O	O	O	O
Base version	O	O	O	O	O

Q4: How important are features for...

	Very important	Important	Neutral	Slightly important	Not important
...delegating the task ('forward')?	O	O	O	O	O
...commenting the task?	O	O	O	O	O

Q5: What would you improve in this view? ____

Question Group #10: EA model visualization (summary)



Q1: The concept of these four visualization layers...

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
… is intuitive.	O	O	O	O	O
...can be helpful for model conflict resolution	O	O	O	O	O
...is helpful for the Enterprise Architect (technical perspective).	O	O	O	O	O
...is helpful for EA Stakeholders (business perspective).	O	O	O	O	O

Q2: Please rate the four layers according to which you deem most useful!
Your choices:
O Schema graph (layer 1)
O Instance list (layer 2)
O Instance neighborhood graph (layer 3)
O Conflict visualization (layer 4)

Q3: Conflict list: As an alternative to the visualizations, we offer a list view for model conflict resolution. The following figure shows four conflicts, each with a link to the conflicting object, a conflict classification, the focal attribute, its values in both information sources to be merged as well as the base value (i.e. the last value both models had in common). Via checkboxes a user can mark the value that shall be adopted into the resulting consolidated model. The button on the right applies the selected value and marks the conflict as solved.

Do you understand the concepts in this view?
O Yes
O No

Q4: This list is...

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
... is intuitive.	O	O	O	O	O
...is helpful for the Enterprise Architect (technical perspective).	O	O	O	O	O
...is helpful for EA Stakeholders (business perspective).	O	O	O	O	O

Q5: What additional information would you need to resolve model conflicts? ______________________________

Q6: What else would you improve in this list? ____

Q7: How important is the conflict list for conflict resolution…

	Very important	Important	Neutral	Slightly important	Not important
...from a technical perspective (Enterprise Architect)?	O	O	O	O	O
...from a business perspective (EA Stakeholder)?	O	O	O	O	O

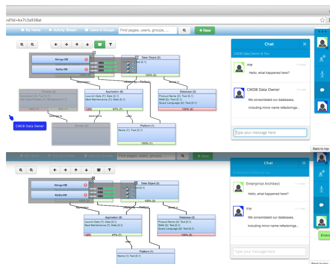
Q8: How important are the visualizations for conflict resolution…

	Very important	Important	Neutral	Slightly important	Not important
...from a technical perspective (Enterprise Architect)?	O	O	O	O	O
...from a business perspective (EA Stakeholder)?	O	O	O	O	O

Q9: Would you prefer the conflict list instead of the conflict visualization?
O Yes
O No

Question Group #11: Collaborative conflict resolution

Q1: Our tool supports real-time collaboration functionality to review and solve conflicts within the visualization. Multiple users can chat, talk, and mutually work on conflicts, similar to the way screen sharing apps work. But as our functionality bases on EA models, different rights and visibilities can be adhered to. In the following example, the user with the role 'CMDB data owner' (bottom screen) has no rights to see the schema elements 'Process' and 'Domain'. The user 'Enterprise Architect' (top screen) is allowed to see everything, yet 'Process' and 'Domain' are greyed out, indicating that his communication partner cannot see them.



	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
The design of these collaborative features is intuitive	O	O	O	O	O
Such collaborative features are useful for conflict resolution.	O	O	O	O	O
The different visibilities due to access rights are useful.	O	O	O	O	O

EA practitioners would easily adopt this means of communication.	0	0	0	0	0
This collaborative functionality would reduce the need for face-to-face meetings.	0	0	0	0	0
I would use the collaborative feature, but my organization is not ready yet.	0	0	0	0	0
Currently e-mail is the way we collaborate.	0	0	0	0	0

Q2: Would you implement different access rights within your EA model?
O Yes
O No

Q3: How often would you use such collaborative functionality?
O Every time I work with the EA tool
O Most of the times I work with the EA tool
O Sometimes when I work with the EA tool
O Rarely
O Never

Question Group #12: Collaborative conflict resolution

Q1: In what kind of industry is your company operating?
O Agriculture, Mining
O Construction
O Education, University
O Finance, Insurance, Real Estate
O Government
O Health Care
O IT, Technology, Internet
O Manufacturing
O Retail, Wholesale
O Services
O Transportation
O Communication, Utilities
O Nonprofit
O Other:_________________

Q2: In which country are you working?
<< Dropdown list with countries provided >>

Q3: Does your company operate on national or international level?
O 1 – 10
O 11 – 50
O 51 – 250
O 241 – 500
O 501 – 1000
O 1001 – 2000
O 2001 – 5000
O 5001 – 10.000
O 10.001 – 50.000
O 50.000 – 100.000
O > 100.000

Q4: What is your present role in your organization?
O Enterprise Architect
O Domain Architect
O Business Architect/Analyst
O CxO
O Developer
O Manager/Head of Department (Business)
O Manager/Head of Department (IT)
O Researcher/Scientist
O Other: _________________

Q5: How many years of experience do you have in EAM?
Only numbers may be entered in this field:______

Q6: How many years has your company been engaged in EAM?
Only numbers may be entered in this field:______

Q7: Does your company have an organizational unit with focus on EA?
O Yes
O No

Q8: Approximately how many employees of your company are permanently working in the field of EAM?
Only numbers may be entered in this field:______