Federated Enterprise Architecture Model Management: Collaborative Model Merging for Repositories with Loosely Coupled Schema and Data

Björn Kirschner

TU München, 85748 Garching, E-Mail: kirschne@in.tum.de

Sascha Roth

TU München, 85748 Garching, E-Mail: roth@tum.de

Abstract

Enterprise Architecture (EA) management seeks for a mutual alignment of business and IT in order to respond to environment changes adequately, e.g. to globalized markets or new technologies. An EA endeavor commonly starts by documenting the current state of the EA in a model. Recent research shows that as of today, EA documentation is time consuming and still regarded highly error prone. Automated EA documentation intends to optimize this crucial step in an EA endeavor. Key to an automated EA documentation is to utilize existing models from operative IT as information sources for EA management, i.e. different models are integrated into an EA model building the topological structure of a federation. In contrast to other disciplines such as software merging or federated database management systems, in many cases user intervention is required to merge EA models in a long-lasting decision process. In this paper, we present ModelGlue, a holistic design for federated EA model management. Our design embraces a meta-meta model, merge facilities, and collaboration support. Through lose coupling of data and schema, the implementation of ModelGlue is more conflict tolerant than other meta modeling platforms. That is, it offers the necessary degree of freedom to serve as a foundation for model integration and collaborative resolution of inconsistencies. For model integration we stress the role of an n-way merge and advocate collaborative as well as configurable conflict resolution strategies.

1 Introduction and Initial Design Considerations Towards Federated EA Model Management

In the last decades, the role of IT has shifted from a mere supporting function through its enabling capabilities towards a game-changing asset of an enterprise. As IT has gained momentum, its management became crucial to successful project execution [22]. Until that time, projects have been executed in a cost-efficient manner limited by the project's horizon. Since every project focused on itself, a holistic management of IT as a whole has been missing for a long time. Consequently, IT has gained complexity and is often considered a substantial part of risk assessment in the course of

enterprise transformations. As a reaction, IT management and respective governance mechanisms took place in organizations. Enterprise Architecture (EA) management seeks to align both business and IT in order to increase flexibility in today's constantly changing environment [22]. Information about an EA is often regarded complex such that specialized EA tools facilitate the respective documentation. As EA management is a young discipline, a plethora of EA tools has emerged [2],[10] in the market over the past decade. Most of these tools are based on an extensible[1] information model. Despite the variety of tools, EA documentation is still regarded very time consuming and error prone. In our recent research, we report on automated EA documentation approaches [15]. The core idea of automated EA documentation is to employ different models distributed in an enterprise to extract respective EA information from runtime models to put it in a central repository; hence, a federated model environment. We assume that due to different terminology, structures, and technology the initial mapping of different information sources to a central EA repository has to be done manually. However, these information sources as well as their models and meta models evolve over time.

Our long-term research objective is to provide means to manage federated models using a central system for their synchronization and maintenance coordination. In the course of synchronization within this federated model environment conflicts are likely to occur. Since we assume an initial mapping between relevant entities of information sources to the central EA repository, changing models (schema as well as data) of information sources and EA repository can be consolidated using an n-way model merge of both schema and data. To resolve conflicts we proposed a conflict resolution process in our recent work [17]. In this work, we continue our efforts towards a federated EA model management. The core idea is to employ tasks and visual means to resolve model conflicts. Thereto, role concepts and conflict visualizations as well as n-way model merges are employed. Next to this high-level design of our solution, we will present a meta-meta model detailing our design of a centralized EA repository in a federated model environment (cf. Figure 1). In line with Di Iorio [5] we advocate a liberal approach to constraint violations. That means we tolerate conflicts on schema, data, and mutual changes of schema and data influencing each other. Against above considerations we conclude to the following research question:

> *How to perform an n-way model merge in a central repository of a federated model environment with loosely coupled schema and data in a collaborative fashion?*

The reminder of the paper is structured as follows: In the next section, we define our notion of federated EA model management. We then give a brief overview of relevant related work. Subsequently, we introduce a meta-meta model as basis for an algorithm design applying an n-way merge of models. Thereby, we focus on an optimistic resolution strategy and the involvement of stakeholders to resolve conflicts between EA models.

2 Federated EA Model Management

Figure 1 shows a typical structure of an automated EA documentation endeavor, cf. [3]. Different linguistic communities use models to describe different universes of discourse. To some extend, however, these universes overlap, as do their respective models. Automated EA documentation provides a central EA repository in a federated modeling environment, which interconnects different information sources to gather information more efficiently [8]. Systems are only integrated partially,

[1] Although the flexibility of the tools varies, for the purpose of this paper, we assume meta-models can be extended with similarly expressive power.

e.g. information about business applications and their interrelationships are extracted from SAP PI (cf. [3]). Integration ranges from screen scrapers, over application layer integration to data integration. As illustrated, information sources may save data in manifold formats, e.g. SQL databases, Excel files, or even from the cloud. Related concepts are synchronized on a regular basis with the central repository. We assume that each information source has a separated presentation of schema and data at runtime; both might change over time. This applies to the central modeling platform, i.e. the EA repository, as well as to the federated information sources.

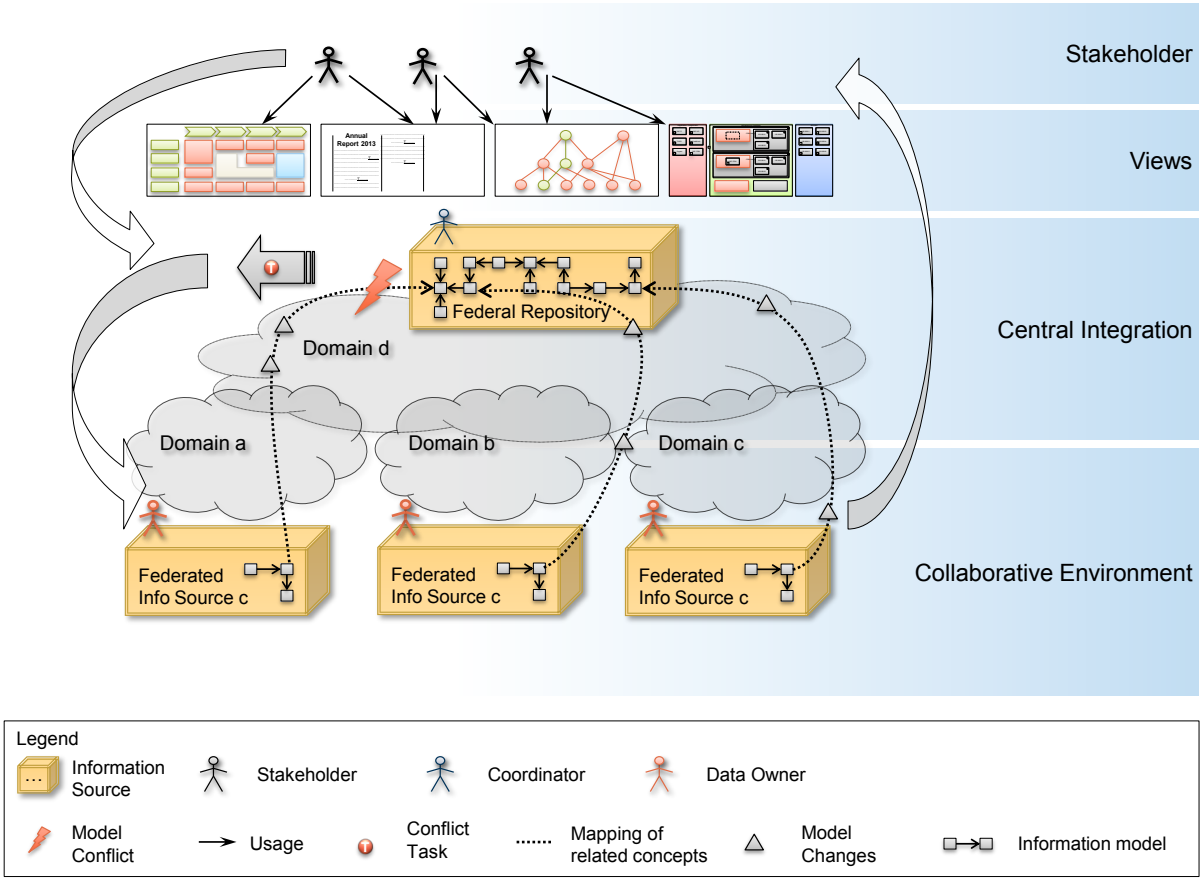

Fig. 1: Conceptual Overview of Federated EA Model Management

3 A Brief Overview of Model Merges in Other Disciplines

Related research is twofold. Some address mere text merging whereas others address the challenge of merging models in the context of model driven software engineering.

Roots of all repository-related functionality for merging are based on textual source code versioning systems. Popular contemporary concurrent versioning systems like Subversion or GIT already offer conflict identification and merge functionality, but only on a line-based textual level. Models, however, can seldom be expressed solely via a text-based structure [23]. Instead, they are often handled at a graph-based abstraction level. Besides that, Rossini et al. [14] allude to the challenge constraints on model elements pose in the event of a model merge. Model merge can be tackled at different levels of complexity: raw, two-way, and three-way merge [4].

Raw merge refers to a merge of two elements with different context. Due to the different context, conflicts are very unlikely and commonly no or very few conflicts are detected [1].

A two-way merge can help to avoid inconsistencies when two contradictory changes to a single object are made simultaneously. A common strategy is to prioritize changes of one replica over the other. With a two-way approach it is impossible to trace changes back to the common anchor, i.e. the original state both versions derived from. If an object exists in only in one of the two models under consideration, it is unclear whether this object has just been created in this model or has been deleted in the other model. This add-delete problem arouses need for baseline data or a historical trace of changes [12].

Due to these shortcomings and the need for additional information, most modern tools perform three-way merging. Therefore, the merging algorithm needs to take the common anchor into consideration [1]. With this baseline information, certain situations can be solved automatically. But if both replicas comprise changes of the very same object, merging still requires human intelligence to avoid arbitrary outcome.

To avoid n-1 pairwise two-way merges when reconciling views of multiple (more than two) stakeholders, Rubin [18] even motivates the need for n-way merging algorithms and introduces a formal framework for this purpose. While common approaches operate on pairs of elements from distinct models, the n-way solutions considers n-tuples of elements [18].

Spanoudakis and Zisman [20] and Finkelstein et al. [7] mention the idea that inconsistencies do not necessarily pose disadvantages. Instead, they can help to create awareness for aspects of the system which possibly need to be refined [5],[13]. Especially Wieland et al. [23] report on turning conflicts into collaboration. In traditional software development, the developer should immediately resolve version conflicts. In contrast, models that are used for knowledge management often incorporate information that may capture concepts in a more informal manner. In EA management, it is essential to develop a common language with the stakeholders and thus conflicts must be 1) tolerated and 2) resolved collaboratively. Wieland et al. [23] use annotations to store model actions of different parties. These can be reviewed and applied to the model as a common understanding thereof evolves via additional communication among stakeholders. At an abstract level, Schmidt et al. [19] take a similar approach. Several publications, e.g. [21], detail the detection and resolution of model conflicts. In contrast to above mentioned approaches we focus on the domain of EA management and propose a solution for separately evolving schemata and data, i.e. to merge co-evolved EA models.

Common EAM tools on the market only provide model-transformation functionality on schema level, not considering schema and data as a whole. As we reported in e.g. [16] and [17], to our best knowledge, only few research addresses conflicts in EA models.

4 The Meta-meta Model

Figure 3 shows a meta-meta model for a repository realizing a loosely coupled schema (EA information model) and data (EA model). In the following, we detail our model with respect to the application domain of EA management. The model element has a unique identifier (uid) and an original identifier (oid). When an existing model element is replicated, i.e. a branch is created, copied elements get a new uid whereas the oid stays as-is. As illustrated in Figure 2, the oid marks the common anchor for branches of models and is precondition for our approach to an n-way merge.

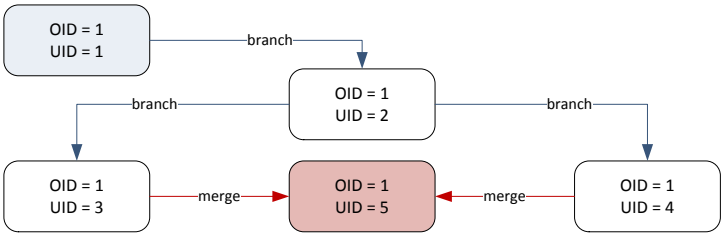

Fig. 2: Original identifier (oid) behavior after branch and merge operations

We separate between objects, attributes, and respective definitions. This realizes the aforementioned loosely coupled schema (object and attribute definitions) and data (object, attribute, and value).

Similar to Farwick et al. [6], roles are attached to a model element in a threefold manner, realizing a fine-grained access control. In addition to explicit read and write access, responsibility is captured as well. Roles may be grouped such that a role can be either a single person or an entire group, possibly with subgroups. In an EA model, it should be possible to define concepts, i.e. models or objects, for which a partially shared responsibility in terms of fine-grained responsibilities is necessary. For example, an object with object definition ‘application’ may have attributes ‘uptime’ and ‘service level’. An instance of such an application, e.g. SAP CRM, has an application owner assigned to it as responsible role for this object. For maintenance, however, responsibility of the attribute ‘uptime’ is

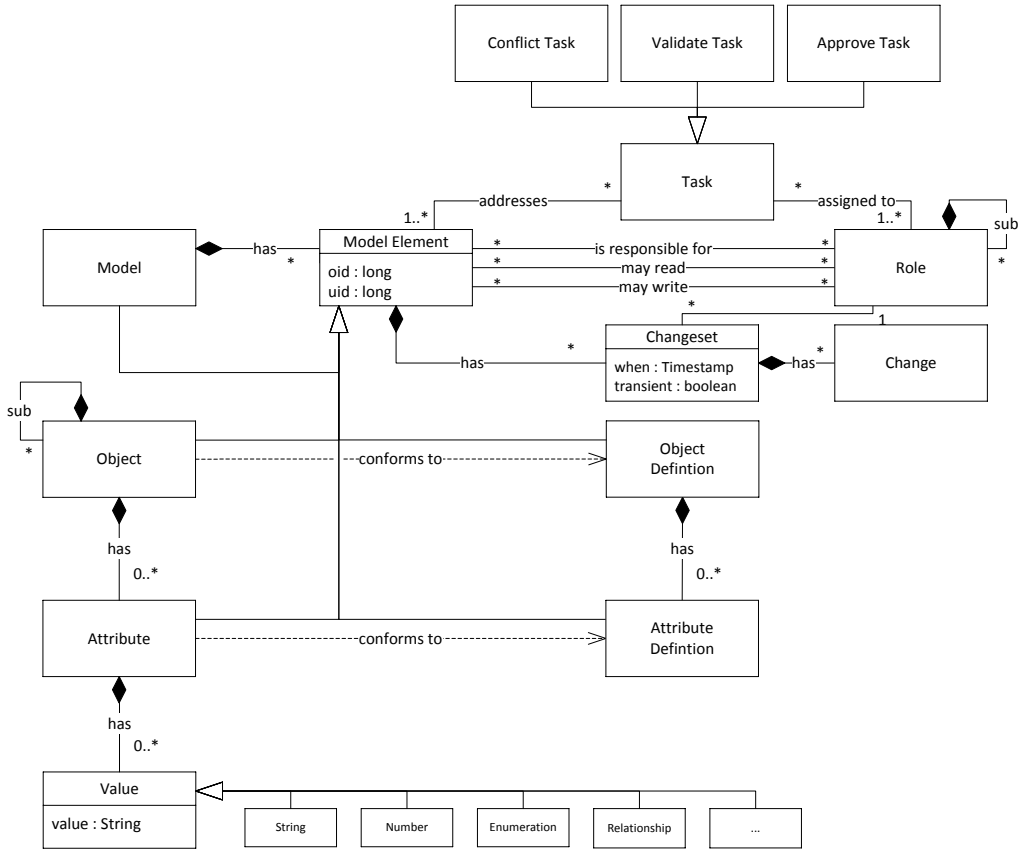

Fig. 3: Conceptual meta-meta model of an EA repository

delegated to the respective system administrator since this role is also responsible for installing patches, etc. When the attribute ‘service level’ of an ‘application’ is switched from value ‘gold’ to ‘silver’, the responsible role assigned to the value ‘gold’ has to be informed. In contrast to Neubert

[13] and through the introduction of the model element, we define access rights and responsibilities on instance level for models, objects, attributes, and values and on schema level for object definitions and attribute definitions. Again, this offers a fine-grained access control as well as more precision when modeling responsibilities.

In line with Neubert [13], our approach offers flexibility, as attributes may or may not conform to an attribute definition. Due to a separation of the concepts attribute, attribute definition and value, end users are allowed to store a value that does not conform to its attribute definition. This is perceived as a larger degree of freedom in an early modeling phase [11]. Inconsistent states of the model in terms of conformance to the respective definitions are thus tolerated by the model and end users may maintain attributes regardless of type conformance. This fosters a bottom-up approach as described in [13]. Neubert calls the explicit specification of an attribute type within an attribute definition type constraint. By default, end users may add attributes relevant for their particular purposes without any attribute definition. These are called free attributes. In the EA domain, a so-called EA repository manager may add an attribute definition based on the frequency an attribute has been used within a model.

We introduce the notion of tasks to semi-automate the model maintenance within our system in order to foster model consistency during model evolution. Thereto, we exploit the access concept of a model element to implement specialized model maintenance tasks. We distinguish between *conflict*, *approve*, and *validate* tasks.

Conflict tasks are created, whenever changes in two or more elements cannot be resolved automatically. One or more responsible end users then will have to decide between them. An example for such a case would be two users having changed the same attribute *a* of an object, such that the value of *a* in branch one differs from the value of *a* in the other branch. As the merge algorithm cannot resolve this situation, a conflict task is triggered to notify the users involved and facilitate a collaborative, manual solution.

Approve tasks allow responsible users to reconfirm conflicting delete actions, which potentially impact the EA model considerably. The distinction between conflict and approve tasks bases on the reasoning that the latter always comprises a delete operation as the dominant change. Approving or disapproving this delete action poses the main objective of the task. For instance an enterprise architect might have deleted an object of the type ‘Apache web server’, while in another branch someone added an attribute called ‘next planned maintenance’ to this object. Merging these two branches raises the question of whether to keep the ‘Apache web server’ object (with the new attribute) or approve its deletion and discard the attribute changes. The attribute change is called non-dominant, as it cannot be applied without first deciding on whether or not to approve of the deletion change. Applying the attribute change requires that the object exists, i.e. the delete action must be disapproved if the attribute change should be applied.

Validate tasks try to detect situations where changes of one user might entail an unintended alteration of the semantics modeled by another user and thus distort the resulting EA model. These tasks allow a specified role (e.g. all writers) to review the changes and possibly correct the semantics of the EA model. A scenario therefor would be one architect modeling 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. In contrast to a conflict task, both changes could be applied here (assuming that no additional constraints had been modeled).

Changesets of a model element are used to keep track of recent changes with the respective role that initiated them. Possible model conflicts are stored within a task including the respective model element, which has access to transient changesets[2].

5 N-way Merge of Co-Evolving Enterprise Architecture Models

Based on the concepts outlined above, we introduce an algorithm to merge models and detect conflicts. The algorithm in Figure 4 illustrates an n-way merge for models based on the meta-meta model illustrated in Figure 3. It returns a list of tasks for each task type (conflict, approve, validate) when given a list of models to merge. These resulting lists comprise the key, i.e. the uid of the focal model element, the common anchor, and a list of conflicting changesets.

Lines 6-9: The algorithm first collects all changesets added after baseline time t_b. Note that the *modelElement* is identified using the oid, i.e. the common anchor. For the sake of brevity, we denote the *modelElement* of a changeset in the following with *c.e*.

Lines 11-22: In a second step, operations taken in different models get consolidated. Thereto all operations performed in the different models must be taken into account.

Line 12: In contrast to text merging, model differences and mutually influencing factors could potentially be distributed. Conflicts may arise between model elements, which are connected via a relationship, e.g. one element contains a reference to another one. As our model requires elements of the same type bearing unambiguous names, the loop must search for these cases as well. Furthermore, links between schema and data need to be considered. For instance, objects could be created already specifying a type without the existence of a type definition. In our repository (cf. Figure 3) the type of an object can be specified using a string. A look-up of an object definition that is named after this string realizes the loose coupling between schema (object definition) and data (object); we apply the same mechanism to attributes and respective definitions. In the event of a concurrent creation of an object definition and an object in different model branches, a mapping of object to object definition takes place implicitly. Given different modelers perform these actions, it cannot be guaranteed that the resulting conformance mapping and respective semantics are intended.

Line 13: A matrix with the four dimensions c_1*.type,* c_1*.operation,* c_2*.type, and* c_2*.operation* stores information about how to classify the conflict. Figure 5 shows the data/data part of the user interface for this conflict classification matrix. Every cell may contain logic assessing relevance and preferred handling of this conflict. A look-up in the conflict classification matrix returns the type of the conflict.

Lines 16-22: We foresee three types of situations during our merge strategy: conflicts, required approvals, and required validations. Depending on the result of the detection routine, changesets are appended to the respective task list. The function *appendChangesetsToTaskList()* guarantees that the newly found conflict between c_1*.e* and c_2*.e* will be appended to the list entry of c_1*.e*, if any conflicts related to c_1*.e* have already been found.

[2] For a detailed description how to save model changes as patches, we refer the interested reader to Kelter et al. [9].

```
     Data: Model M_{1..n}, Baseline Time t_b, Target Model M_t
     Result: Task T_{1..n}
 1:  conflicts ← (key : ∅, value^{t_b} : ∅, value : {∅})
 2:  approve ← (key : ∅, value^{t_b} : ∅, value : {∅})
 3:  validate ← (key : ∅, value^{t_b} : ∅, value : {∅})
 4:  changesets ← {∅}
 5:  // 1. collect operations
 6:  foreach element ∈ M_{1..n} do
 7:      foreach changeset ∈ element do
 8:          if changeset.when > t_b then
 9:              changesets ← changesets ∪ changeset
10:  // 2. consolidate operations
11:  foreach c_1 ∈ changesets do
12:      foreach c_2 ∈ changesets : ∃relationship(c_1.e, c_2.e)
         ∨(c_2.e.type ≡ c_1.e.type ∧ c_2.e.name = c_1.e.name) ∨ c_2.e.instanceof(c_1.e)  do
13:          conf = classificationMatrix[c_1.type][c_1.op][c_2.type][c_2.op]
14:          // Lookup in Conflict Classification Matrix
15:          if conf ≠ ∅ then
16:              switch conf do
17:                  case conflict
18:                      conflicts ← appendChangesetsToTaskList(conflicts, c_1, c_2)
19:                  case approve
20:                      approve ← appendChangesetsToTaskList(approve, c_1, c_2)
21:                  case validate
22:                      validate ← appendChangesetsToTaskList(validate, c_1, c_2)
23:  foreach e ∈ (conflicts.keys ∪ approve.keys ∪ validate.keys) do
24:      foreach c ∈ changesets do
25:          if e ≡ c.modelElement then
26:              changesets.remove(c)
27:  // 3. apply to target model
28:  // 4. check consistency/schema conformance
29:  // 5. user intervention
30:  // 6. create conflict resolution tasks
```

Fig. 4: Merge Algorithm

Lines 23-26: When detection and classification is completed, every changeset involving conflicts will be removed from the list of changsets. Thus, in a subsequent step each non-conflicting change can be applied to the target model automatically.

Before actually conducting a merge, we offer an overview of model changes that will be performed during a merge scenario to the end user. At this point, we first perform a consistency check, which on the one hand employs the loose binding of information model (schema) and model (data) mentioned above and on the other hand checks for possible constraint violations. Subsequently, the end user can review the scenario, modify possible conflict situations, and finally apply the merge. Based on the merge and previously made analysis of conflicts, responsible roles of model elements get notified via tasks (cf. Figure 1).

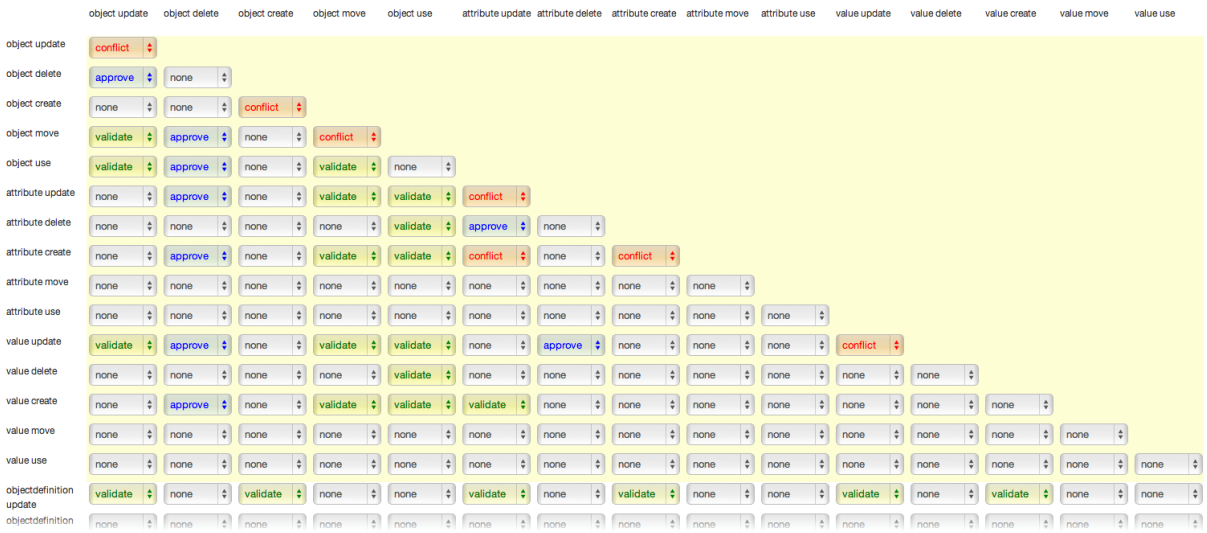

Fig. 5: Conflict Classification Matrix in the ModelGlue tool (data/data part)

Detection and Classification of Conflicts

Whenever a potential conflict is found, the conflict classification matrix (cf. Figure 5) assesses and classifies it and returns the appropriate task type (i.e. conflict, approve, or validate). Apart from these three standard types, ModelGlue offers the option to define custom behavior for every cell using a simple expression language. Figure 5 shows the pre-defined strict detection strategy, which, however, may be customized to allow more tolerant resolution strategies. In the following, the reasoning behind some typical cases will be explained via examples.

Case: create object definition / create object definition: Two users both create an object definition specifying the same name. Our model, however, requires names of object definitions to be unambiguous. As the function cannot decide which object definition to keep, it returns a *conflict*, which can later be solved manually via user involvement.

Case: update object / delete object: User u_1 updates an object while another user u_2 deletes this very object in another branch; note that both objects have the same oid marking a common anchor in their model of origin. Now it is unclear whether the deleting user u_2 would still adhere to his erasure considering the updated information provided by u_1. So an *approve* task is triggered. A person responsible for the object under consideration has to resolve the issue.

Case: move object / create attribute: For instance, an object of type 'JBOSS application server' gets redeployed to run an 'Apache web server'. As one user moves the object into the 'Apache web server' container, another one simultaneously adds a new attribute 'Java version' to the object, as he is not aware of the change. A responsible role now has to decide whether the updates performed on the object are still valid and sensible within the new context, i.e. if it is sensible and necessary to store the 'Java version' under the new 'Apache web server' context. Hence, a *validate* task is returned.

Case: create value / update attribute: In one branch, a user adds a second value to an attribute whereas in another branch the cardinality of this very attribute gets restricted to 0..1 ('at most once'). In this case, a *validate* task must ascertain whether to keep the new value or not. This way, we allow a certain degree of inconsistency and thus freedom in early modeling stages, but strive to refine the model quality incrementally by resolving such issues via validate tasks.

Case: update object definition / create attribute: As already mentioned before, our system facilitates loose coupling between schema and data. Therefore object definitions per default allow all objects implementing it to add additional free attributes. Those are attributes which are not specified in the object definition. The property '*allow free attributes*', however, may also be set to *false* in order to force objects to strictly adhere to their object definition. If a user u_1 adds a free attribute to an object not being aware of user u_2 concurrently setting '*allow free attributes*' to false, a *validate* task is triggered to resolve the issue.

6 Conclusions

The contribution of this paper is an n-way merge algorithm, which detects model conflicts and generates resolution tasks. This algorithm is based on the presented meta-meta model for federated EA model management. In contrast to EAM tools and merge approaches in other disciplines such as model-driven software engineering, we regard schema and data as a whole, addressing potential problems on both levels as well as conflicts between schema and data.

The generated tasks provide means to reach consistency in the federation. ModelGlue, the implementation of our approach, employs a repository with loosely coupled schema and data to offer a necessary degree of freedom to discuss conflicts and reach model consistency. We advocate that especially for EA management, in many cases user interaction is essential to resolve complex model conflicts in a long-lasting and highly collaborative process. Particularly schema level conflicts must be resolved by modeling experts closely collaborating with different parties. Often this is crucial for the success of EA management since EA initiatives commonly depend on stakeholder buy-in. In this vein, we favor (temporarily) storing inconsistencies over loosing information. Our vision is to reach an eventually consistent state in all models within the federation.

As a next step, concepts and implementation of ModelGlue will be evaluated in the industry, assessing relevance, acceptance and technical correctness. At the MKWI 2014 student track we intend to showcase our implementation of the approach to gather feedback and team up with other research groups.

7 Literature

[1] Barrett, S; Chalin, P; Butler G (2008): Model merging falls short of software engineering needs. *Proc. of the 2nd Workshop on Model-Driven Software Evolution.*

[2] Berneaud, M; Buckl, S; Diaz-Fuentes, A; Matthes, F; Monahov, I; Nowobliska, A; Roth, S; Schweda, CM; Weber, U; Zeiner, M (2012): Trends for enterprise architecture management tools survey. Technical report, Technical University Munich.

[3] Buschle, M; Ekstedt, M; Grunow, S; Hauder, M; Matthes, F; Roth, S (2012): Automated enterprise architecture documentation using an enterprise service bus. *Proceedings of in Americas conference on Information Systems (AMCIS)*.

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

[5] Di Iorio, A; Draicchio, F; Vitali, F; Zacchiroli, S (2012): Constrained wiki: The wikiway to validating content. *Advances in Human-Computer Interaction*.

[6] Farwick, M; Pasquazzo, W; Breu, R; Schweda, CM; Voges, K; Hanschke, I (2012): A meta-model for automated enterprise architecture model maintenance. In: Chi, CH; Gasevic, D; Heuvel, WJ (editors), *EDOC*, pages 1–10. IEEE.

[7] Finkelstein, ACW; Gabbay, D; Hunter, A; Kramer, J; Nuseibeh, B (1994): Inconsistency handling in multiperspective specifications. IEEE Transactions on Software Engineering, 20(8):569–578.

[8] Hauder, M; Matthes, F; Roth, S (2012): Challenges for automated enterprise architecture documentation. In: Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation, pages 21–39. Springer.

[9] Kelter, U; Kehrer, T; Koch, D (2013): Patchen von Modellen. In: Software Engineering 2013.

[10] Matthes, F; Buckl, S; Leitel, J; Schweda, CM (2008): Enterprise Architecture Management Tool Survey 2008. Technical report, Technical University Munich.

[11] Matthes, F; Neubert, C; Steinhoff, A (2011): Hybrid wikis: Empowering users to collaboratively structure information. *Proceedings of the 6th International Conference on Software and Data Technologies (ICSOFT)*.

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

[13] Neubert, C (2012): Facilitating Emergent and Adaptive Information Structures in Enterprise 2.0 Platforms. PhD thesis, Technical University Munich, München, Germany.

[14] Rossini, A; Rutle, A; Mantz, F; Lamo, Y; Wolter, U (2010): Constraint-aware model merging. *Proceedings of NWPT*, pages 31–33.

[15] Roth, S; Hauder, M; Farwick, M; Breu, R; Matthes, F (2013): Enterprise architecture documentation: Current practices and future directions. *11th International Conference on Wirtschaftsinformatik (WI)*, Leipzig, Germany.

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

[17] Roth, S; Hauder, M; Matthes, F (2013): Facilitating conflict resolution of models for automated enterprise architecture documentation. *Americas Conference on Information Systems (AMCIS), 2013*.

[18] Rubin, J; Chechik, M (2013): N-way model merging. *9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

[19] Schmidt, M; Wenzel, S; Kehrer, T; Kelter, U (2009): History-based merging of models. *Comparison and Versioning of Software Models, 2009. CVSM '09. ICSE Workshop*, pages 13–18.

[20] Spanoudakis, G; Zisman, A (2001): Inconsistency management in software engineering: Survey and open research issues. Handbook of software engineering and knowledge engineering, 1:329–380.

[21] Taentzer, G; Ermel, C; Langer, P; Wimmer, M (2010): Conflict detection for model versioning based on graph modifications. *Proceedings of the 5th international conference on Graph transformations, ICGT'10,* pages 171–186, Berlin, Heidelberg, 2010. Springer-Verlag.

[22] Weill, P; Ross, JW (2009): IT Savvy: What Top Executives Must Know to Go from Pain to Gain. Harvard Business Press.

[23] Wieland, K; Langer, P; Seidl, M; Wimmer, M; Kappel, G (2012): Turning conflicts into collaboration - concurrent modeling in the early phases of software development. Computer Supported Cooperative Work: The Journal of Collaborative Computing, tba: pages 1–52.