Towards Metamodel Integration Using Reference Metamodels

Johannes Meier
Software Engineering Group
University of Oldenburg, Germany
meier@se.uni-oldenburg.de

Andreas Winter
Software Engineering Group
University of Oldenburg, Germany
winter@se.uni-oldenburg.de

ABSTRACT

The complexity of modern software engineering projects increases with growing numbers of artefacts, domain specific languages, and stakeholders with their concerns. To overcome these demands, different viewpoints are used to describe different languages specifying different artefacts, specific concerns of stakeholders, and domain specific languages. Therefore, the use of different viewpoints together in one software engineering project increases and requires technical support for automatic synchronization of the used viewpoints. This paper gives an overview about use cases for viewpoint synchronization and compares their fulfillment by existing approaches. As result, this vision paper proposes a new approach for synchronization of viewpoints to overcome the presented use cases with focus on reduction in synchronization and integration effort, on reuse of integration knowledge, and on viewpoint evolution.

1. MOTIVATION

Viewpoint-oriented software engineering is becoming more and more an essential paradigm in modern software engineering. It allows different viewpoints on the current system under development, and is motivated by an increasing number of different artefacts and languages, which are involved in modern software systems. Working with one artefact written in one language out of several means using one viewpoint out of several viewpoints pointing on the same information of the system.

As an *example* to describe and develop software, viewpoints for representing requirements in textual form, designing required static data in form of UML class diagrams, object-oriented implementation using Java, and for testing with JUnit testcases could be used. They all are working on information, which form together the domain. The term *domain* describes all relevant information of the current project. This example stems from the domain of object-oriented software development (OOSD) and is an ongoing one through the complete paper.

VAO '16 March 2, 2016, Karlsruhe, Germany

DOI: 10.475/123_4

Different viewpoints support different concerns of different stakeholder with *tailored viewpoints*. Following the definitions of ISO 42010:2011 [8], a *viewpoint* determines selected concepts of the domain addressing specific concerns. A *view* contains a subset of the concrete information of the domain on instance level and is conform to one viewpoint.

An important property of viewpoints on technical level is the possibility for *overlapping viewpoints* [7]. This allows using same concepts in different viewpoints. As an example, the tester needs at least reading access to the source code, and to the requirements which are giving specifications what to test. This concept allows several stakeholders to work together at the same software project using different viewpoints. Overlapping viewpoints raise the problem of duplicated data which are changed through different viewpoints which is a source for inconsistencies, which leads to the need for *synchronization* of data for being consistent [6].

Working with several viewpoints requires also, that further information on content level "between" the viewpoints can be expressed. As an example, relations between the viewpoints for requirements and Java source code specify, which part of the source code fulfills which requirement. To support this kind of traceability between viewpoints on content level, new viewpoints could be defined for this purpose.

These aspects have to be realized on technical level, before user are able to work in viewpoint-oriented way. While working, the user needs support to synchronize viewpoints with other ones. Because of this different use cases and their actors, this position paper prepares several use cases in viewpoint-oriented projects as first step (Section 2). After comparison with existing approaches, which shows different fields for improvements, this position paper introduces a new approach for the synchronization of viewpoints for user, and for reduced integration effort for methodologists (Section 3). The approach will also support existing viewpoints and corresponding existing data, the evolution of that integration by the methodologists, and the reuse of integration effort in future projects by methodologists. This position paper ends with a conclusion and an outlook in Section 4.

2. USE CASES

The two most important stakeholder in viewpoint-oriented projects are the *user* who uses one or more viewpoints for working with different artefacts, and the *methodologist* [1, 15] who creates and manages the viewpoints and their relationships with each other. In the following, several use cases for user and methodologists are described and their fulfillment by existing approaches discussed.

2.1 Integrate existing Viewpoints

Initially, the methodologist integrates existing viewpoints somehow. Suitable existing approaches from literature are divided into *synthetic* and *projectional* approaches, following the ISO for Architecture Description 42010:2011 [8]. These two approaches are visualized in Fig. 1 using the ongoing example. Synthetic approaches keep the views of the different

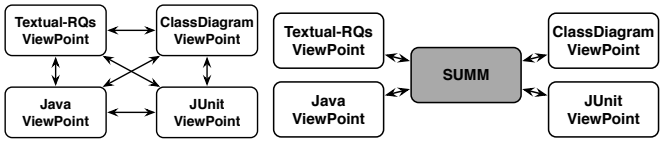

Figure 1: Synthetic vs. Projectional Approaches

viewpoints unchanged and independent from each other, and require some kind of synchronization between all viewpoints for consistency (Fig. 1). Main characteristic is the *pair-wise synchronization* between the viewpoints to propagate changes from one viewpoint into all other viewpoints. To synchronize overlapping concepts like classes of viewpoints for Java and class diagrams, synchronizations directly between these two viewpoints are introduced.

The several *synthetic* approaches differ from each other by using different techniques for synchronizations: *Triple Graph Grammars* (TGGs) are used for specifying bidirectional relations between different graph languages [13]. In opposite to that, [12] uses *QVT-R* for synchronization.

Projectional approaches integrate the existing viewpoints into one singe *integrated metamodel* (later called SUMM) which contains the concepts off all viewpoints in an integrated manner, whereby synchronization is done only between viewpoint and SUMM. Each viewpoint works with a subset of the concepts of the SUMM, and propagates changes in the view to the central model. In Fig. 1, the central metamodel called SUMM contains all concepts of the viewpoints for textual requirements, UML class diagrams, Java source code, and JUnit test cases. Duplicated concepts like classes in the viewpoints for Java source code and UML class diagrams are only once in the SUMM which saves pair-wise synchronization effort. Instead, the viewpoints containing a subset of the concepts of the SUMM propagate changes in their views into the SUM, which forwards the changes to the other views which contain also these changed concepts. There are also several *projectional* approaches which differ from each other mostly in the underlying techniques.

Orthographic Software Modeling (OSM) [1] uses a so called Single Underlying Model (SUM) which contain all information about the current domain. Therefore, a metamodel for the SUM is needed, the Single Underlying MetaModel (SUMM). The SUM will be changed only through changes in views on it, respectively viewpoints on the SUMM. The views will be synchronized with the SUM by transformations. An environment for OSM is realized prototypically [2]. The issue, how to get a SUMM, is marked as future work [2].

The *Vitruvius* approach [10] follows the projective OSM ideas of a SUM with views on it, but implements this idea internally by a so-called modular SUM (MSUM). For that, the existing models are kept independent from each other like in synthetic approaches (without an explicit SUMM), and are combined with relations between each pair of viewpoints expressed through a new DSL called MIR, from which synchronization transformations are derived. In the end, on technical level, Vitruvius can be seen as synthetic approach.

2.2 Import and integrate existing Data

If in viewpoint-oriented projects data already exist which are conform to the integrated viewpoints, then these data have to be imported and reused. In particular, if data are coming from different overlapping viewpoints, not only the viewpoints have to be integrated, but also the instance data. Additionally, existing tools use models which conform to fixed metamodels of viewpoints. As result, the approach has to keep the existing viewpoints as first viewpoints to allow import and export for existing data and tools. This use case is motivated also in Vitruvius [10].

An advantage of synthetic approaches is, that existing viewpoints are directly usable, because only additional mechanisms for synchronization between the existing viewpoints will be created. Therefore, no additional effort is required to keep existing viewpoints and views usable for existing tools. Keeping existing viewpoints is significant easier in synthetic approaches, because projectional approaches take the existing viewpoints and integrate them into a SUMM. The presented projectional approaches in the modeling area do not give hints, how to deal with existing viewpoints.

2.3 Create new Viewpoints

In approaches for viewpoint-oriented engineering, methodologists have to create new viewpoints to support new external tools with a fixed viewpoint and new concerns of stakeholders. Important is, that new viewpoints have to use all aspects of the current domain [6]. In particular, concepts of several existing overlapping viewpoints together with their interrelations have to be reusable for new viewpoints. This allows easy creation of new viewpoints while reusing the elaborate work for integration.

Reusing concepts of different viewpoints is hard in synthetic approaches, wherefore Vitruvius introduces a declarative DSL called ModelJoin [3]. With ModelJoin, new viewpoints can be defined declaratively using concepts of several viewpoints, while the required metamodel of the new viewpoint and the synchronization transformations will be generated from the declarations. For the projectional approaches, [4] has similar ideas like the OSM approach and focuses on how to create new viewpoints on basis of a SUMM. For that, [4] developed an editor which helps to create new viewpoints as subsets of the SUMM. The needed synchronizations between new viewpoints and the SUMM will be generated, which use model differences for propagating changes between SUM and all views.

2.4 Synchronize Views

After integrating viewpoints and importing existing data, user read and write all data through initial and new viewpoints. Changed concepts in one viewpoint have to be synchronized into all viewpoints containing these concepts to keep overlapping viewpoints consistent.

General problem of synthetic approaches is the *square number of relations between the viewpoints*, which synchronize overlapping views of overlapping viewpoints to avoid inconsistencies. This results in heavily increasing initially creation effort. By contrast, in projectional approaches the number of required synchronizations is linear in the number of viewpoints. This is achieved by deleting duplicated concepts in the SUMM. At runtime, changes in a view are synchronized into the SUM, and from there forwarded into other views which also contain the changed elements.

2.5 Create a similar Integration again

Another problem is the initial effort for methodologists to integrate all viewpoints. While this initial effort is in general not avoidable, the reuse of previous done integration in future projects simplifies viewpoint integration. As an example, in a future project, requirements, Java code, and testcases should be used like before, but now Extended Entity Relationship (EER) diagrams should be used for describing static data. It would be nice to reuse the integration of requirements, source code, and testcases, and exchange class diagrams through EER diagrams easily. This results in the problem, how to reuse integration knowledge in future projects in the same domain. Ideas for *reusing integration knowledge* inside domains are not mentioned in the presented related work, which shows field for optimization.

2.6 Evolve Viewpoints

After finishing the integration of viewpoints, the initial viewpoints and their integration evolve because of, as examples, new versions of the tool specifying the viewpoint, or new version of Java like in the ongoing example. It is important, that evolution is possible, all unrelated metamodels remain the same, and that the co-evolution of existing models will be handled [6]. Neither the presented synthetic nor projectional approaches support currently this use case.

3. NEW APPROACH

This section proposes a new approach for viewpoint integration. Summarizing the related work for the different use cases in viewpoint-oriented projects, the integration and synchronization of projectional approaches has linear effort compared with square effort for synthetic approaches, while projectional approaches lack in handling existing data. Because this limitation is removable with manageable effort (compare Section 3.2), and the creation of new viewpoints is possible in both approaches but easier in projectional ones, the new approach of this position paper extends the projectional OSM approach. The following text show, how this new approach look like and will fulfill all the use cases.

3.1 Integrate existing Viewpoints

To create the SUMM which have to contain the concepts of all viewpoints exactly once, the metamodels of the viewpoints will be taken and integrated into one big metamodel, the SUMM. This integration will be done by the methodologist who has to define on semantic level, which concepts are duplicated in several viewpoints, and which additional relations between concepts have to be added. On technical level, several operators will by applied on the metamodels in a step-wise way. Besides typical operators like `add`, `change`, and `delete` [11], some more specialized refactorings like merging two classes representing the same concept into one single class are required. The selection of appropriate operators is part of future work. These operators allow the integration of the metamodels of the different viewpoints into one single underlying metamodel which is required by the OSM approach as input.

3.2 Import and integrate existing Data

An open issue in the existing OSM approach is, how to handle existing data which are conform to the integrated metamodels. This is important, because existing tools, for example, for modeling UML class diagrams, should be used together with the new approach. As an example, existing Java source code should be used further on. This means in both cases, that existing viewpoints and their data (here for UML class diagrams and for Java source code) have to be usable together within the new approach. On technical level, the concrete metamodels have to be kept as viewpoints on the final SUMM through the complete integration process.

This should be ensured by the in Section 3.1 proposed step-wise execution of operators on the metamodels with parallel creation of transformations for the model-co-evolution [5]. After integrating the viewpoints into the SUMM, the corresponding co-evolution-transformations allow the integration and import of the existing data into the SUM. If each step is combined with a complementary operator and a complementary co-evolution-step for the other direction, the data export from the SUMM into the viewpoints will be executable.

3.3 Create new Viewpoints

To create new viewpoints onto an existing SUMM, the existing projectional approach of [4] will be reused: After selecting a subset of the SUMM which forms the viewpoint, the required synchronizations between new viewpoints and the SUMM will be generated.

3.4 Synchronize Viewpoints

Extending the OSM approach allows linear effort for synchronization transformations. For future implementations, approaches for the propagation of model differences between views and the SUM like [15] which is used in the OSM approach or like [4] can be used also in this approach.

3.5 Create a similar Integration again

Looking at the ongoing example, concrete metamodels for Java code and class diagrams are integrated into a SUMM following the OSM approach. If a new project uses C++ instead of Java, the complete integration has to be performed again, while for integration purposes only general concepts like classes and methods are required. To allow such reuse of integration knowledge in the new approach, these general concepts will be moved into reference metamodels (RMMs).

Following [16], reference models are modeling the main and general characteristics of sets of systems of the same kind, and hiding specialized aspects of individual systems [9]. Reference models serve as reference point for specialized models [16] and support the construction of specialized models reusing the concepts of the reference model [14].

Because the specialized models are metamodels for Java and C++, a *reference metamodel* will be created. Java and C++ have both generalizable characteristics like classes and methods, and specialized aspects like different handling of pointers. The generalizable characteristics are part of the reference metamodel for object-oriented programming languages, while the specialized aspects remain in the concrete metamodels for Java and C++ (Fig. 2). The RMM for

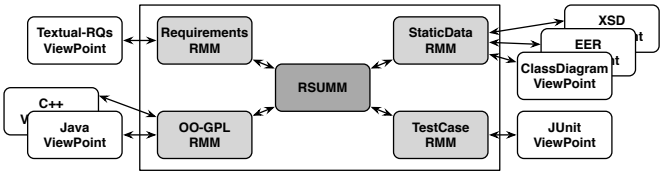

Figure 2: Concepts of the new approach

the subdomain of object-oriented programming languages would contain at least classes containing methods. Concrete metamodels for Java and C++ will be mapped onto their corresponding RMM, whereby as example the details of

the Java and C++ metamodels are ignored on RMM level. The RMMs of all subdomains will be integrated into the RSUMM. The RSUMM contains the integration on conceptual level, like here the integration of classes and methods from object-oriented programming languages, with classes and attributes of data description languages.

In concrete projects, for each subdomain one concrete metamodel will be selected, like Java as programming language and Extended Entity Relationship (EER) diagrams as data description language. Based on the RSUMM, the RMM parts will be replaced by concrete metamodels to get the SUMM, whereby the integration knowledge will be derived from the integration done in the RSUMM. The derived SUMM is the same like in the OSM approach.

This approach allows to integrate the subdomains as representatives for the main concepts only once in form of the RSUMM for each domain, instead of each time in form of SUMMs for each new project. The integration in form of the RSUMM can be reused for each project by "instantiating" each subdomain by the currently needed concrete metamodel to get a SUMM. This allows arbitrary combinations of concrete metamodels and their viewpoints.

Another advantage of the new approach is, that for new concrete viewpoints like XML Schema Definition (XSD), its mapping to the StaticDataRMM is enough, and no complex integration with other subdomains has to be performed, because the integration is done before using the RMM. After creating the mapping, XSD can be used directly.

The technique for mappings between concrete metamodels and their reference metamodels is part of future work, which could apply step-wise metamodel changes (Sec. 3.1).

3.6 Evolve Viewpoints

After finishing the metamodel integration by the methodologist and while user are working with the integrated viewpoints, evolution can occur to the integrated viewpoints: The evolution of new viewpoints (CMMs) basing on the SUMM can be expressed again as metamodel changes together with the creation of model-co-evolution transformations to handle the instance level. The evolution of the CMMs can be simplified in this approach by distinguishing changes into changes which affect only unimportant concepts which are not part of the RMM, and into important changes which affect both the CMM and the RMM. While the former case can be realized as refactoring of the CMM, the latter case requires also changes in the RMM and therefore also in the RSUMM, which will be complex and requires further investigations. But after handling the evolution task in the RSUMM, all derived SUMMs benefit and apply them.

4. CONCLUSION

For technical support of viewpoint-oriented software engineering, the viewpoints have to be integrated by the methodologist to offer the user consistent information across several viewpoints. Therefore, this position paper presented different use cases in viewpoint-oriented projects, and compared their fulfillment by several synthetic and projectional approaches. As result, synthetic approaches have the problem of square synchronization effort, projectional approaches lack in supporting existing viewpoints, and all approaches do not support the evolution of viewpoints and the reuse of integration results in future projects.

To overcome these limitations, this paper proposes a new approach following the OSM approach to have only linear synchronization effort. The new approach extends the OSM approach by support for existing viewpoints, which allows using existing data and reusing existing tools. This is reached by rigorously creating model-co-evolution mechanisms for all metamodel changing steps, which are needed to map concrete metamodels to their reference metamodels and to integrate reference metamodels into the RSUMM.

Main contribution and benefit of the new approach compared to existing approaches is the reuse of integration effort for future projects within the same domain, which is not in the focus of other approaches. This is reached by shifting the integration of concrete metamodels to the integration of concepts expressed in reference metamodels (RMM) containing the main concepts of subdomains. This allows the methodologist to select one of several possible metamodels like for Java or C++ for the current project. As result, the complex integration will be done once on reference level, and will be reused while deriving SUMMs for current projects.

The proposed ideas of this position paper will be concretised and implemented in a framework for viewpoint-oriented software engineering. As domain for validation, the ongoing example of this paper of object-oriented software development will be used, which could be extended by further subdomains like project management, or documentation.

5. REFERENCES

[1] C. Atkinson, D. Stoll, and P. Bostan. Supporting View-Based Development through Orthographic Software Modeling. *Enase*, pages 71–86, 2009.

[2] C. Atkinson, D. Stoll, C. Tunjic, and J. Robin. A Prototype Implementation of an Orthographic Software Modeling Environment. VAO 2013.

[3] E. Burger, J. Henss, M. Küster, S. Kruse, and L. Happe. View-based model-driven software development with ModelJoin. *Software & Systems Modeling*, 2014.

[4] A. Cicchetti, F. Ciccozzi, and T. Leveque. A hybrid approach for multi-view modeling. *Recent Advances in Multi-paradigm Modeling*, 50, 2011.

[5] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. *12th EDOC'08*, 2008.

[6] R. France and B. Rumpe. Model-driven Development of Complex Software: A Research Roadmap. *FOSE 2007*.

[7] T. Goldschmidt, S. Becker, and E. Burger. Towards a Tool-Oriented Taxonomy of View-Based Modelling. 2012.

[8] IEEE. ISO/IEC/IEEE 42010:2011 - Systems and software engineering - Architecture description. 1–46, 2011.

[9] H. Krallmann and B. Scholz-Reiter. Cim-KSA - Eine Rechnergestützte Methode für die Planung von Cim-Informations- und Kommunikationssystemen. *Informatik-Fachberichte*, 258:57–66, 1990.

[10] M. Kramer, E. Burger, M. Langhammer. View-centric engineering with synchronized heterogeneous models. *VAO'13*.

[11] D. Kuryazov and A. Winter. Representing Model Differences by Delta Operations. *EDOCW 2014*.

[12] J. R. Romero, J. I. Jaén, and A. Vallecillo. Realizing correspondences in multi-viewpoint specifications. *EDOC 2009*.

[13] A. Schürr and F. Klar. 15 Years of triple graph grammars: Research challenges, new contributions, open problems. *Lecture Notes in Computer Science*, 5214:411–425, 2008.

[14] O. Thomas. Understanding the Term Reference Model in Information Systems Research: History, Literature Analysis and Explanation. *LNCS*, 3812 (Chapter 45):484–496, 2006.

[15] C. Tunjic and C. Atkinson. Synchronization of Projective Views on a Single-Underlying-Model. *VAO 2015*.

[16] A. Winter. *Referenz-Metaschema für visuelle Modellierungssprachen*. Deutscher Universitäts-Verlag, 2000.