Metamodel-Compliance Checking of Requirements in a Semiformal Representation

Hermann Kaindl[1], Stefan Kramer[2], Mario Hailing[3], and Vahan Harput[3]

[1] Vienna Univ. of Technology, Inst. of Computer Technology
Gusshausstr. 27-29, A-1040 Vienna, Austria
kaindl@ict.tuwien.ac.at
[2] Technische Univ. München, Inst. f. Informatik/I12
Boltzmannstr. 3, D-85748 Garching b. München, Germany
[3] Siemens AG Österreich, PSE, Geusaugasse 17, A–1030 Vienna, Austria

Abstract. Checking requirements is highly desirable but hard to achieve in practice, where only word processors are used in most projects. While in this case reviews are more or less the only means, formal representations allow for a variety of automated checks. Unfortunately, formal representations of requirements are rarely available in real-world projects. *Semiformal* representations are easier to obtain and still offer some possibilities for automated checks. Based on an object-oriented hypertext representation, we present an implemented approach for compliance checks against a *metamodel*.

1 Introduction and Background

In this paper, we present an approach to *metamodel-compliance checking* that is based on a semiformal representation. A given model can be said to be *compliant* with a *metamodel* if all parts prescribed by the metamodel are contained in the model.

We have implemented this approach to metamodel-compliance checking in the tool RETH (Requirements Engineering Through Hypertext) that supports an object-oriented hypertext representation of requirements. These representations are semiformal in the sense that they can be partly processed by machine based on their form. For example, a hypertext link can be followed, and an attribute can be *inherited* in a class-subclass structure. Making use of the formally represented parts, this tool can check compliance against a metamodel.

RETH's basic hypertext approach is described in [1], based on which requirements, goals and domain entities are all modeled as objects in RETH. Classes and instances are distinguished, and they are described in one hypertext node each in the tool. In this way, descriptions in natural language of, e.g., requirements statements or glossary entries are integrated in the object-oriented structure. The hypertext nodes in RETH have also internal structure through so-called *partitions*, which can represent object attributes, for example.

Additionally, RETH denotes a method supported by this tool. The approach to representing requirements can be found in [2], and a process of scenario-based requirements engineering in [3]. In the context of this paper, the metamodel defined by this method is important, since it provides the basis for the metamodel-compliance checks. As given

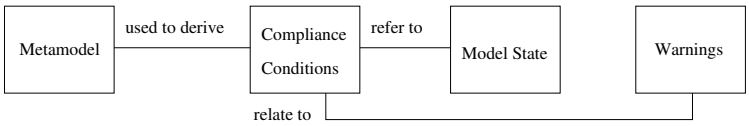

Fig. 1. Conceptual overview of our metamodel-compliance checker.

in [2–4], it defines which objects are to be represented and which relations between objects. For example, goals and scenarios are to be represented as well as a many-to-many relationship between them.

The remainder of this paper is organized in the following manner. First, we present our approach to metamodel-compliance checking. After that, we sketch some empirical evidence based upon a student project. Finally, we relate our approach to previous work.

2 Our Approach to Metamodel-Compliance Checking

Since our approach focuses on compliance with the RETH metamodel, the checker compares the concrete requirements information (a given semiformal model in the RETH tool) with this metamodel. Fig. 1 shows a conceptual overview of this metamodel-compliance checker. More precisely, it checks certain *compliance conditions* that can be derived from this metamodel. For example, the metamodel prescribes that scenarios (more precisely scenario instances) should exist. The derived compliance condition says that at least one scenario instance should be present in the given model. As a more interesting example, the metamodel prescribes that scenarios and goals should be in a many-to-many relation. From this, two compliance conditions can be derived: one says that for each already existing scenario instance at least one goal instance should exist to which it is connected according to this relation, the second one is analogous in the other direction.

The user can invoke checking these compliance conditions at any time, either selectively or all of them together. There are two ways to configure the checker for a given application:

1. Whenever part of the metamodel is discarded, e.g., by leaving out scenarios, the checker works with the remaining part only.
2. While focusing the current work on a specific part, e.g., on defining a domain model, all the unrelated checks can be switched off selectively and temporarily.

Our tool does not enforce strict conformance to the metamodel at all times, it allows a model to be temporarily non-conformant. This is particularly important in a metamodel as presented above, where e.g., for each scenario at least one goal is to be represented and vice versa. Since such objects can only be entered one after another, strict conformance would not be feasible at all times.

The metamodel-compliance checks simply result in warnings if something appears to be missing. It should be clear that the metamodel-compliance checks involved here can merely indicate *potential* problems. The user decides whether such a potential problem is really a problem in the given context of using RETH. Therefore, our checker

issues warnings to the users in order to make them aware of potential problems they might either overlook or forget about.

An obvious compliance condition derived from the metamodel is that instances of functional requirements should be specified. If the checker invokes it and cannot find a single one, it issues a respective warning. Missing instances of quality requirements, scenarios and goals are handled analogously. Also the domain model should not be empty. There will even be a warning if only object instances exist and no single object class. In addition, the checker issues a warning, if descriptive text is missing in certain attributes. The more complicated conditions derived from the relation between scenarios and goals were sketched above.

3 Empirical Evidence

Now let us sketch empirical evidence from a student project, where the participating computer-science students had the opportunity to use the RETH tool to specify requirements for a Web-based game-playing software system. The goal was to find evidence whether

1. the RETH checker helps creating models compliant with the RETH metamodel;
2. the RETH checker is easy to use and improves the overall acceptance of the tool.

We had two homogeneous groups of eight students each. One group used the tool with the RETH checker, one without. We handed out a problem statement, and we obtained requirements documents from each participant. We then carefully reviewed each document according to a list of criteria related to the metamodel compliance and the contents of the documents.

In addition, we asked all participants to fill out a questionnaire containing closed questions regarding the usefulness and usability of the tool and its features, with an additional line for free form comments. In general, we designed the questionnaire according to [5, p. 208–212]. In particular, we used a multi-point rating scale, more precisely a *Likert scale*.

Regarding the aspects monitored by the RETH checker, its use resulted in models with higher compliance, at least as compared with no checking by machine at all. Interestingly, the checker seems to improve other aspects of metamodel compliance as well. One possible explanation might be that users pay more attention to other aspects of metamodel compliance by analogy. In addition, the results from the questionnaires reflecting the subjective opinions of the users in this student project are in favor of our approach and its supporting tool. In particular, the progressive assistance was well accepted by the users.

4 Related Work

In contrast to an abundance of approaches based on formal representations, recently a more lightweight approach to consistency checking was proposed building on an object-oriented semiformal representation [6]. It deals with consistency between a scenario model and a class model.

We found, however, much less work on *completeness* checking in the literature. Again, formal representations prevail, primarily used for checking completeness of an implementation with regard to a specification. While the focus of [6] is on consistency as stated above, it also deals with a kind of *partial incompleteness*. Our approach to metamodel compliance can also be viewed as completeness checking of the structure of a model, whether it contains all the parts prescribed by the metamodel.

The closest work in the literature deals with *standards compliance* [7]. Standards prescribe practices, which not only include certain procedures, but also constraints that must hold for documents. Such a standard is, however, less rigidly defined than our metamodel. Even if more formally defined in a tool like DOORS as described in [7], standards compliance would usually not prescribe all the parts of a model in the same way as a metamodel.

5 Conclusion

We have presented a novel approach to compliance checking against a metamodel, more precisely the RETH metamodel. This approach is implemented in the RETH tool for checking whether a requirements specification in the tool contains the parts that the method prescribes. While the scope of this work was metamodel compliance of requirements, the approach may well be more generally applicable, whenever a metamodel is specified for what is to be represented in a corresponding tool.

In summary, the contributions of this paper are

- metamodel-compliance checking based on a semiformal representation,
- progressive assistance allowing a model to be temporarily non-conformant, and
- embedding of this approach in a hypertext tool for requirements engineering.

6 Acknowledgments

The Forschungsförderungsfonds für die gewerbliche Wirtschaft (FFF) in Austria supported part of this work under contract 800489.

References

1. Kaindl, H.: Using hypertext for semiformal representation in requirements engineering practice. The New Review of Hypermedia and Multimedia **2** (1996) 149–173
2. Kaindl, H.: A practical approach to combining requirements definition and object-oriented analysis. Annals of Software Engineering **3** (1997) 319–343
3. Kaindl, H.: A design process based on a model combining scenarios with goals and functions. IEEE Transactions on Systems, Man, and Cybernetics (SMC) Part A **30** (2000) 537–551
4. Ebner, G., Kaindl, H.: Tracing all around in reengineering. IEEE Software (2002) 70–77
5. Dumas, J.S., Redish, J.C.: A practical guide to usability testing. Ablex Publishing Corp., Norwood, NJ (1993)
6. Glinz, M.: A lightweight approach to consistency of scenarios and class models. In: Proceedings of the Fourth International Conference on Requirements Engineering (ICRE2000), Schaumburg, IL, IEEE (2000) 49–58
7. Emmerich, W., Finkelstein, A., Montangero, C., Antonelli, S., Armitage, S., Stevens, R.: Managing standards compliance. IEEE Transactions on Software Engineering **25** (1999) 836–851