Sketching a Model-Based Technique for Integrated Design and Run Time Description

Short Paper - Tool Demonstration

Andreas Kästner, Martin Gogolla, Khanh-Hoang Doan and Nisha Desai

Computer Science Department, University of Bremen, Bremen, Germany
{andreask|gogolla|doankh|nisha}@informatik.uni-bremen.de

Abstract. The paper sketches a UML- and OCL-based technique for the coherent description of design time and run time aspects of models. The basic idea is to connect a design model and a run time model with a correspondence model. We show two simple examples, one for structural modeling and one for behavioral modeling, that explain the underlying principles. As all three models are formulated in the same language, UML and OCL, one can reason about the single models and their relationships in a comprehensive way.

1 Introduction

In recent years, design time (DT) and run time (RT) models of software as well as their interplay have become a major topic in research and development. Often, it is said that the advantage of using a model instead of code lies in its power to abstract away unnecessary technical details. But up to now, a common agreement about the distinguishing characteristics of DT and RT models and their relationship is still open. This paper proposes to formulate an explicit DT and an explicit RT model and to formally link both.

For expressing these models, we use a mainstream language, the UML (Unified Modeling Language) [8], which includes the OCL (Object Constraint Language) [10]. Our approach is explained with examples that are worked out in our tool USE (Uml-based Specification Environment)[1].

As said, to catch DT and RT modeling aspects, we propose to introduce three connected models as sketched in Fig. 1: (a) a design time model, (b) a run time model, and (c) a correspondence model that connects and constrains the first two models. All three models can be full models containing, e.g., classes, associations, and constraints, but a model may also consist of associations (as first-class citizens) and constraints only. The correspondence model depends on and imports the other two models. All interactions and dependencies between design time and run time are modeled here. Figure 1 displays in the upper part the three contributing models. In order to be more concrete, an example is given in the lower part with classes, associations and constraints for the structural modeling example to be discussed in detail further down.

[1] https://sourceforge.net/projects/useocl/

The paper layout follows the LNCS style.
Only the white margins have been reduced.

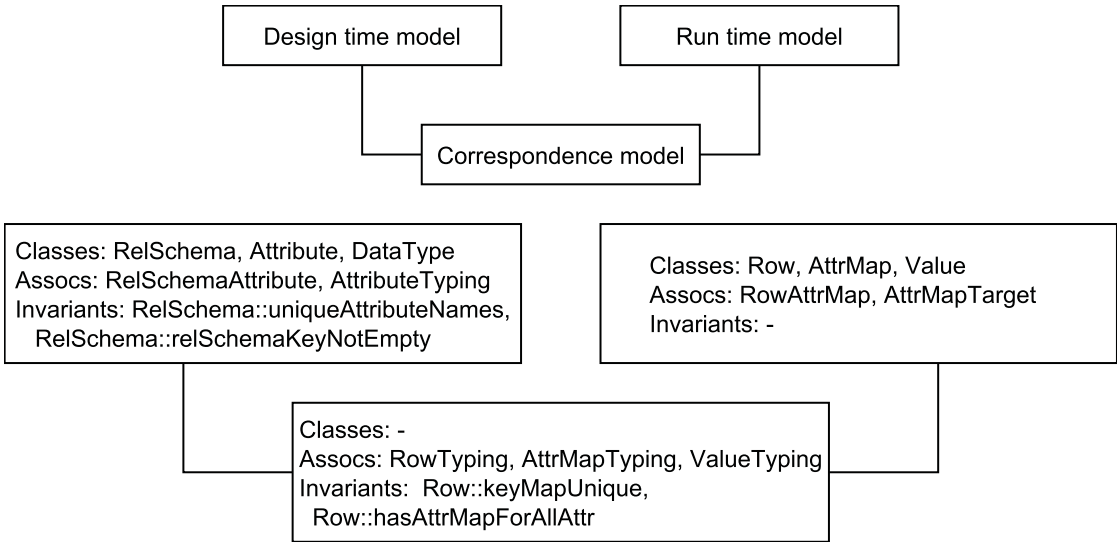

Fig. 1. Design time, run time and correspondence model.

The research contribution of this paper lies in the proposal for the distinction of the three different models and in the proof-of-concept that it is possible to realize this structure in a software design tool. We are not aware of another proposal for a correspondence model. The advantage that we see in such an explicit model lies in the option to analyze the relationship between DT and RT model, e.g., to check RT errors and to trace and to identify the 'guilty' parts either in the DT model or in the RT model or in both models.

Within USE a so-called monitor [5] is available that allows to introduce a RT model: a running application can be monitored in terms of a UML and OCL model and thus the behavior and possible constraint violations of the application can be checked. For the presented simple examples, we have not yet used the monitor, but have constructed the RT model explicitly.

The rest of this contribution is structured as follows. Section 2 introduces the structural example model. Section 3 focuses on the behavioral example model. Both examples show a RT error in form of an invariant violation, and both are implemented in USE [3, 4]. Section 5 discusses some related approaches. Section 6 ends the paper with concluding remarks and future work.

2 Structural Modeling Example

The class model in Fig. 2 shows a tiny SQL subset: (a) in the DT model on the left, we see that a relational schema (class `RelSchema`) has attributes and that an attribute is typed through a data type; (b) in the RT model on the right, a relational schema is populated with rows in which each attribute gets a value by means of attribute map objects; (c) the correspondence model consists of three typing associations that allow to connect the RT objects with a unique type.

Further rules are stated in the form of invariants that restrict the possible the object models. The names of these invariants are shown in the Class invariants window. We informally explain the constraint purpose in the order in which the invariants appear: (a) the set of key attributes of each relational schema has to

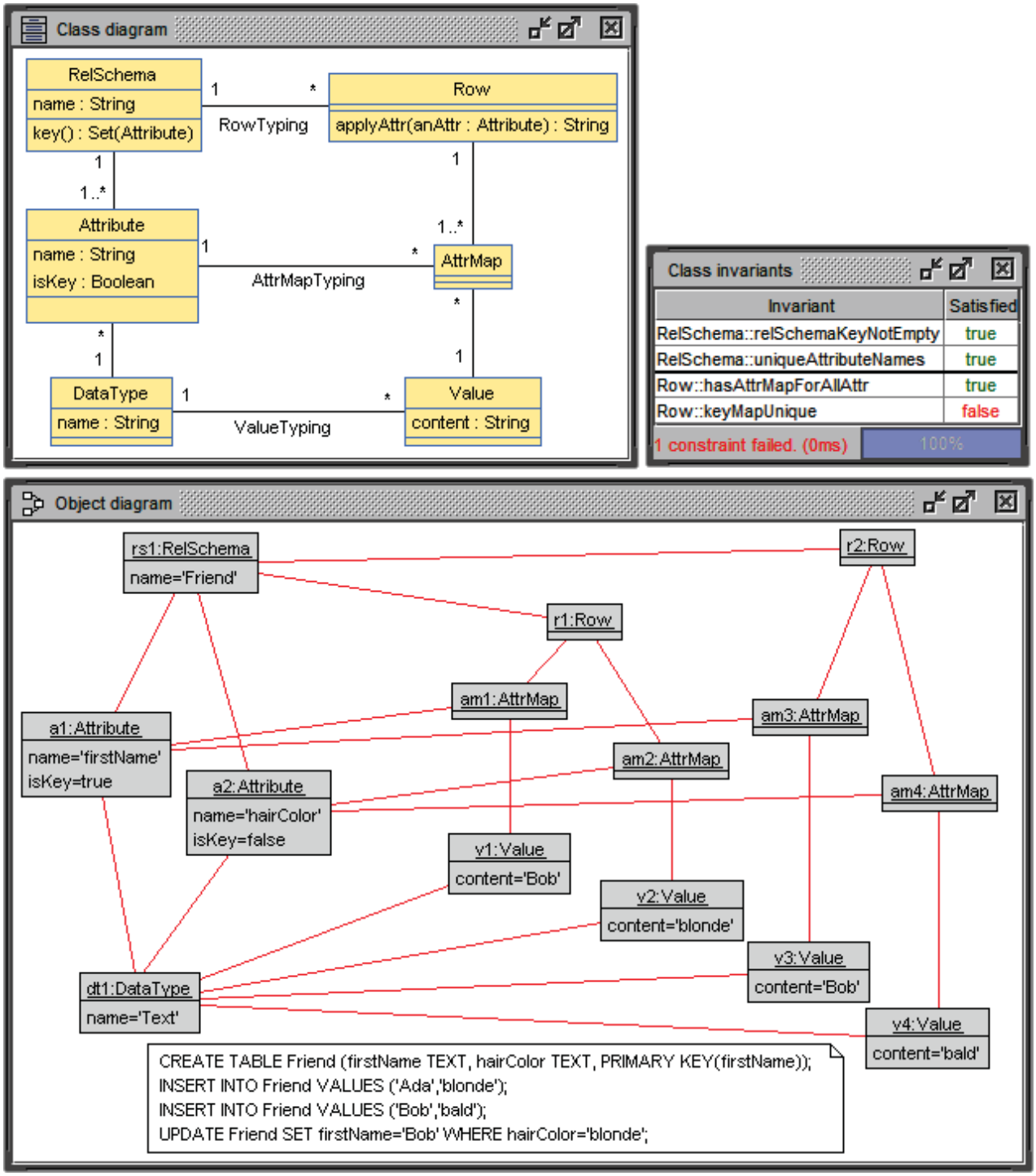

Fig. 2. DT, RT and correspondence elements for a relational database.

be non-empty, (b) the attributes names have to be unique within the relational schema, (c) each row must have an attribute value for each of its attributes, and (d) each row must have unique key attribute values.

In the lower part of the figure, we see a usage scenario in concrete SQL syntax. One table (relational schema) is added with a `create` command, populated by two SQL `insert` commands and finally modified with an additional SQL `update` command. This usage scenario is represented in the form of an evolving object model. The figure shows only the last object model after the SQL `update` has been executed: (a) after the `create` command only the four left-

most objects (rs1, a1, a2, dt1) are present; (b) after the first insert command the five middle objects (r1, am1, v1, am2, v2) appear, however we will have v1.content='Ada'; (c) after the second insert the five right-most objects (r2, am3, v3, am4, v4) will appear; up to this point all four invariants evaluate to true; (d) after the update command the content value of v1 changes (v1.content='Bob') and the evaluation of the invariant keyMapUnique turns to false. This constraint violation corresponds to a RT error that is indicated to the developer and that can be analyzed further with our tool so that the Value object v1 is identified as being 'guilty' for the RT error. In this example, the correspondence model consists of associations and invariants only, but one could think of more complicated situations with RT objects introduced at different points in time and having different DT types (e.g., osama:TaxPayer and osama:Terrorist). This could be reflected by a correspondence class and appropriate objects.

3 Behavioral Modeling Example

Figure 3 shows a DT and RT model for simple protocol state machines. In the class model on the left side, we have the class State and the association class Transition making up the DT model. On the right side, TraceNode and TraceEdge constitute the RT model. The association between State and TraceNode establishes the correspondence model.

The class model is illustrated by an object model that instantiates in particular the DT and RT classes. The object model pictures an automatically generated [4] fitness example. It shows in the middle a protocol state machine with states named BOILING, FREEZING, and COLD as well as transitions labeled jog, run and stretch. This instantiates the DT model. In the left and in the right of the class model, two examples traces, i.e., executions of the protocol state machine, instantiate the RT model: the actual event sequences are in the first execution on the left {stretch; run} and in the second execution on right the sequence {jog; run; stretch} and through links belonging to the correspondence model, the TraceNode objects are connected to State objects.

In the upper right part of Fig. 3 the names of needed OCL invariants are presented: (a) the OCL invariants for the DT part require deterministic transitions, each state to lie between the initial and the final state, unique state names, and the existence of a single initial and a single final state; (b) the OCL invariants for the RT part require each trace to be a cycle-free string of pearls; (c) the invariants for the correspondence part demand each trace to be connected to the initial state and the traces to show events corresponding to transition events.

The invariants in particular check that the sequence of events from the two traces is correct traces from the specified protocol state machine. In this case, the right event sequence {jog; run; stretch} is an acceptable sequence, however the left event sequence {stretch; run} is not a sequence allowed by the protocol state machine. This leads to the observation that the invariant TraceEdge::eventTraceEdge_EQ_eventTransition evaluates to false: the link between TN6

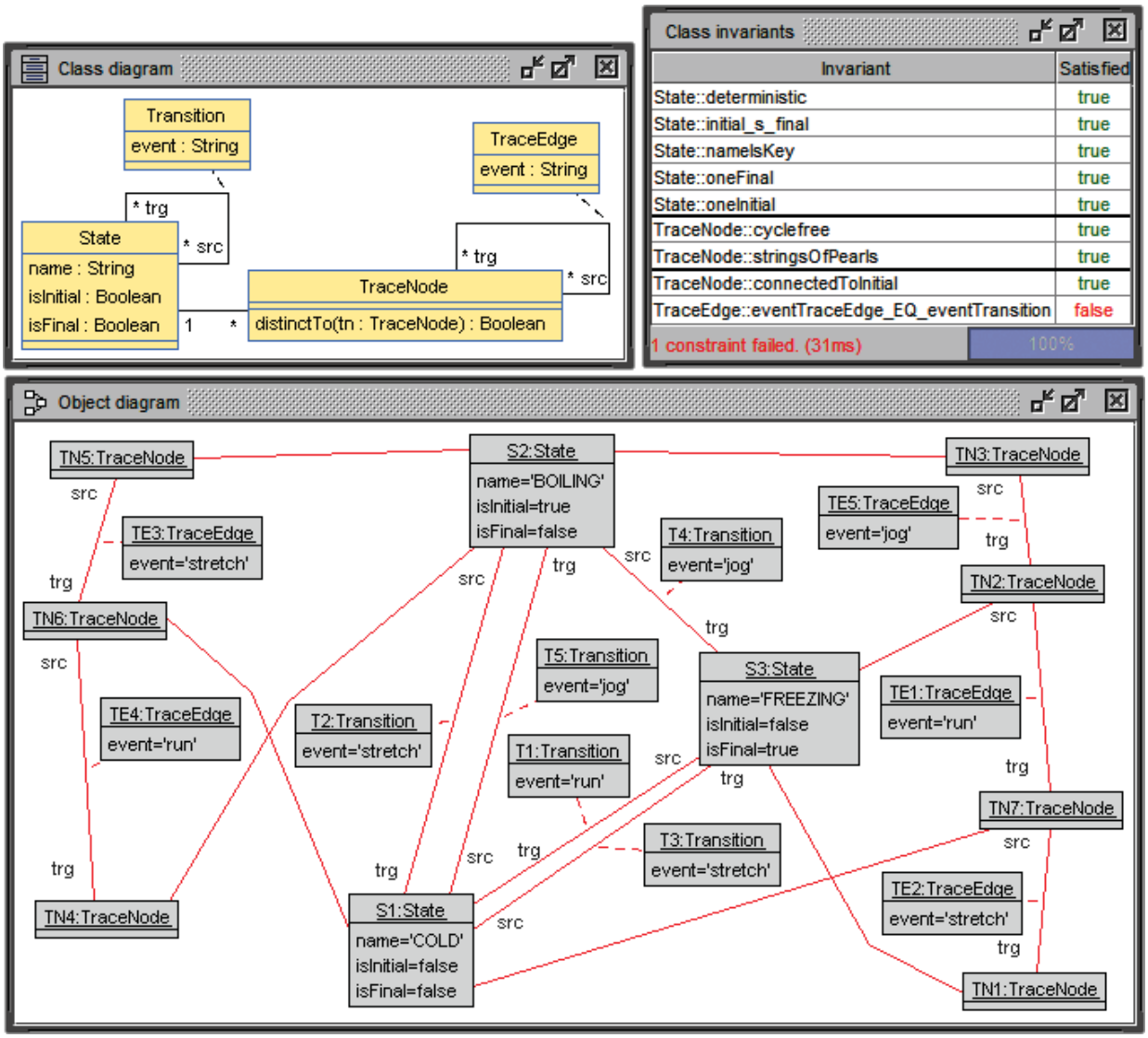

Fig. 3. DT, RT and correspondence elements for a protocol state machine.

and `TN4` violates the determined protocol. Our tool USE offers options in terms of a so-called evaluation browser to analyze the object model and to identify the source for invariant violation: in the example the `TraceNode` objects `TN6` and `TN4` could be brought into the foreground.

4 Related Work

In [2], the authors propose an approach for improving user interaction modeling by adopting a design uncertainty model into an IFML model. Uncertainty is then solved by integrating the results of a run time log analysis. The approach in [1] discusses the Requirements Modeling Languages (RML) and proposes a conceptual distinction between design time and run time requirements models. Run time models extend design time models with additional information about execution of system tasks. In [7], the authors introduce an aspect-oriented modeling approach to enhance software adaptation by unifying design time and run time adaptation. [6] gives an overview on run time verification specification languages. [9] discusses through a controlled experiment whether it helps for comprehension of run time phenomena when corresponding design time models are provided.

5 Conclusion

The problem discussed in this contribution has been to formulate the connection between a design time and a run time model in a coherent way. We have shown by two examples how to use a software design tool to represent a connecting correspondence model. Future work includes finding a general way to set up the structure of the correspondence model. One may also introduce schematic, template-based correspondence models that establish unique typing connections from the RT model to the DT model. Tool support must be extended in order to formally distinguish between the different models. Experiments for constructing run time models with the USE monitor should be carried out. Last but not least, larger case studies and examples should check the applicability and usefulness of the proposed technique.

References

1. Borgida, A., Dalpiaz, F., Horkoff, J., Mylopoulos, J.: Requirements models for design- and runtime. In Atlee, J.M., Baillargeon, R., Chechik, M., France, R.B., Gray, J., Paige, R.F., Rumpe, B., eds.: Proc. 5th Int. Workshop on Modeling in Software Engineering (MiSE 2013), IEEE Computer Society (2013) 62–68
2. Brambilla, M., Eramo, R., Pierantonio, A., Rosa, G., Umuhoza, E.: Enhancing flexibility in user interaction modeling by adding design uncertainty to IFML. In Burgueño, L., et al., eds.: Proc. MODELS 2017 Satellite Events. Volume 2019 of CEUR. (2017) 435–440
3. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Journal on Science of Computer Programming, Elsevier NL **69** (2007) 27–34
4. Gogolla, M., Hilken, F., Doan, K.H.: Achieving Model Quality through Model Validation, Verification and Exploration. Journal on Computer Languages, Systems and Structures, Elsevier, NL (2017) Online 2017-12-02.
5. Hamann, L., Vidacs, L., Gogolla, M., Kuhlmann, M.: Abstract Runtime Monitoring with USE. In Mens, T., Cleve, A., Ferenc, R., eds.: Proc. European Conf. Software Maintenance and Reengineering (CSMR'2012), IEEE (2012) 549–552
6. Havelund, K., Reger, G.: Runtime verification logics A language design perspective. In Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., Mardare, R., eds.: Models, Algorithms, Logics and Tools. LNCS 10460, Springer (2017) 310–338
7. Parra, C.A., Blanc, X., Cleve, A., Duchien, L.: Unifying design and runtime software adaptation using aspect models. Sci. Comput. Program. **76**(12) (2011) 1247–1260
8. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language 2.0 Reference Manual. Addison-Wesley, Reading (2003)
9. Szvetits, M., Zdun, U.: Controlled experiment on the comprehension of runtime phenomena using models created at design time. In Baudry, B., Combemale, B., eds.: Proc. ACM/IEEE 19th Int. Conf. MODELS, ACM (2016) 151–161
10. Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley (2003) 2nd Edition.