A Master Level Course on Modeling Self-Adaptive Systems with Graph Transformations

Matthias Tichy
Organic Computing, Department of Computer Science
University of Augsburg, Augsburg, Germany
tichy@informatik.uni-augsburg.de

ABSTRACT

A growing emphasis in software and systems development is being laid on the integration of self-x characteristics like self-healing, self-adaption, self-optimization. More often than not those characteristics can be modeled in terms of graphs and graph transformations. At the Organic Computing group at the University of Augsburg we addressed the topic of modeling self-adaptive systems in a semester long course at the master level. It was designed to contain extensive practical applications of software modeling tools to go along with the lectures. We employed GROOVE and Fujaba as tools for the practical assignments w.r.t. to graph transformations. We report about the course topics, the practical assignments and lessons learned.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design

Keywords

Self-X, model-driven development, graph transformations, teaching, Fujaba

1. INTRODUCTION

The current trend for the increasing embedding of software in technical systems is accompanied by requirements that these technical systems should be working as good as possible by optimizing itself and adapting to changes in the environment or the users needs.

This requirement adds additional complexity to the already complex distributed software in todays embedded systems. Model-driven development approaches are employed to counter this growing complexity by abstracting from details of the hardware platform or the underlying middleware as far as possible. Model-driven approaches are often built on domain specific languages which are tailored to the problems and needs of a specific domain.

Kramer and Magee [10] suggest to adapt and to use the three layer architecture of Gat [5] for self-adaptive systems consisting of the following layers: (1) *goal management*, (2) *change management*, and (3) *component control*. The component control layer does contain the architectural configuration of the self-adaptive systems, i.e. the components and their connections that are active in a certain configuration. Besides the execution of the components, this layer is responsible for the execution of reconfiguration plans. These plans, which consist of actions like adding, removing, and replacing of components and connectors, are stored in the change management layer and are executed in response to events. The plans are computed in the goal management layer and correspond to goals.

The graph transformation formalism is a natural fit for the specification of structural self-adaption as envisioned by Kramer and Magee. A number of different approaches based on graph transformations have been developed in the past [17, 11, 19, 8] including our own [18] for the modeling of self-adaptive systems.

In the master level course "Modeling self-adaptive systems" at the University of Augsburg in the winter term 2010/2011, we took the challenge to give an overview about model-driven approaches for the development of self-adaptive systems. The lectures contain material about all parts of the three layer architecture. So, the students did learn about components and architectures, the specification of behavior for the components, specification of reconfiguration actions with graph transformations, automated planning for the computation of reconfiguration plans, and requirements for self-adaptive systems. We did focus on the aspect of modeling reconfigurations using graph transformations. We did employ GROOVE and Fujaba as tools. The course was attended by nine students as the master program at the University of Augsburg as well as the Organic Computing chair is rather new.

Sections 2 and 3 contain a presentation of the topics for the lectures and the practical assignments, respectively. We conclude with a summary of lessons learned and an outlook on future work in Section 4.

2. LECTURE TOPICS

There is currently no standard approach for the development of self-adaptive systems even less a standard text book which can be used in lectures because this research area is new and addresses a wide variety of problems. Additionally, we intended to provide the students with lecture topics that they can also use in non self-adaptive systems.

Therefore, we decided to build the lecture on the following three themes being conscious of the fact that these themes only cover a small part of the current research on self-adaption: (1) model-driven development, (2) architectural/structural approaches to self-adaption, and (3) practical experience with modeling tools.

The lecture was divided into the following content blocks:

Introduction In the introduction, we reviewed several examples of technical systems which include a heavy amount of software as e.g. current airplanes. We finally did take a look at the autonomous vehicles of the RailCab-project at the University of Paderborn. The software of these autonomous vehicles exhibits many characteristics of self-x systems like self-healing, self-optimizing, self-adaption. We employed the RailCab-project as a running example for the lectures.

Definitions This content block begins with an introduction to modeling and model-driven development. The main part of this lecture deals with the different terms which are used in the research community like self-adaption, organic computing. As there are currently no standard definitions for this term, several different definitions were given and discussed with the students.

Architecture One of the main themes of the lectures is self-adaption by architectural reconfiguration. Therefore, we reviewed in this content block definitions of architecture, configuration and components as well as architecture description languages. Our focus was laid on the architectural patterns for self-adaptive systems. This includes open and closed control loops as well as patterns as the aforementioned three layer architecture and the MAPE-K architecture [13].

Graph transformations We focus in the lecture on the structural adaptation as one kind of self-adaptation. We introduced graph transformations as basic formalism for the specification of single structural changes. As specific graph transformation formalism we introduced GROOVE [14], Story Diagrams [3] and Component Story Diagrams [18]. The last one is a variant of Story Diagrams which specifically targets architectural reconfiguration in self-adaptive systems.

Automated Planning According to the three layer architecture, plans are an ordered set of single actions which fulfill a certain goal. Automated planning (cf. [7]) is the discipline which targets the computation of such plans. The Planning Domain Definition Language (PDDL) [4] is a textual modeling language for planning problems. In this content block of the lecture we introduced the PDDL as well as different planning algorithms. Graph transformations can be easily mapped to actions in the PDDL with the exception of node creation and deletion. Therefore, automated planning approaches can be integrated with graph transformations.

Automata / Statecharts In this content block, we introduced automata, timed automata and Statecharts and their varying semantics for the model-driven development of the state-based behavior of components in self-adaptive systems.

Requirement Languages Finally, we did take a short look on requirements languages for self-adaptive systems which focus on the inherent uncertainty of self-adaptive systems. Specifically, we did take a look at RELAX [20].

The lecture can be extended by content blocks about quantitative analysis for self-adaptive systems like stochastic petri nets or probabilistic automata for availability and reliability analysis.

3. PRACTICAL ASSIGNMENTS

The lecture topics are accompanied by practical assignments for groups of two students. We decided to use a tool-centered approach for the practical assignments as we believe that students are better motivated when using tools and trying to get their solutions working than doing pen and paper assignments – if the tools work.

After a presentation of the running example, we describe those practical assignments which deal with graph transformations. We keep to simple examples due to space constraints.

3.1 Running Example

We use a self-healing distributed system as a running example throughout the practical assignments. The example is inspired from distributed embedded systems as e.g. in automotive systems. Currently, neither self-adaption nor self-healing is employed in current automotive systems but envisioned in current research approaches [1, 12, 9].

The distributed system consists of a hardware layer and a software layer. The hardware layer is built of nodes, called electronic control unit (ECU), and busses which connect an arbitrary number of nodes. The software layer consists of a set of component types which need to be instantiated on the ECUs in order for the system to be operating. Each component type may be available in different variants. Component types additionally specify required connections between their instances. The corresponding connections of the component instances are called links.

Finally, component variants and connections require certain types of resources (e.g. RAM) from an ECU or a bus, respectively. EUCs and busses provide those resources. We refrained from using real hardware and kept the scenario to simple simulations. This allowed us to concentrate on the modeling part and not to mess with the additional complexity of embedded hardware and software.

We employed other examples as e.g. the famous elevator and ferryman examples in the practical assignments as well. They are much simpler than the self-healing scenario and therefore provide an easier introduction to the different modeling formalism and corresponding tools.

3.2 Modeling the structure

The first assignment consisted of modeling the structure of the self-healing system. Figure 1 shows the resulting class diagram.

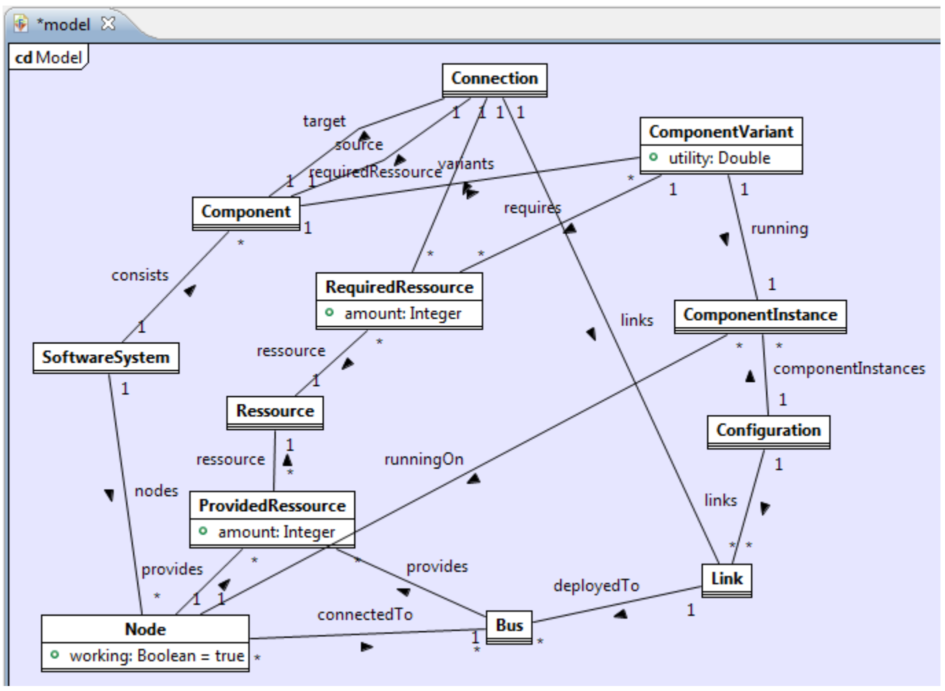

Figure 1: Class diagram of the self-healing example.

A second part of the structural modeling deals with a simulation environment. The simulation environment is event-driven, i.e. it contains a sorted set of events. The events are sorted with respect to the time the events should be executed. Examples of events are failures of component instance, busses and nodes as well as self-healing actions.

3.3 Modeling self-healing actions

Self-healing actions in the running examples are starting and stopping of component instances, repairing of nodes and busses as well as connection the component instances. The practical assignments were initially restricted to stopping and starting of component instances.

We initially used GROOVE as the tool for the specification and execution of self-healing actions as it enables an easy and fast way to introduce graph transformations to new people. We switched to Fujaba after the students were familiar with graph transformation in order to employ its code generation facilities and tight integration with Java.

Figure 2 shows a story diagram which models the creation of a new instance of a component. The story pattern selects a running node and instantiates an arbitrary component variant. Additionally, a failure event with a trigger time is already created and added to the event queue.

In addition to the self-healing actions, the simulation environment contains sensors which periodically measure system properties like the system's availability and output them to files for post-processing (e.g. plotting).

3.4 Modeling a planning problem

The three layer architecture groups single actions into plans which are executed in order to reach goals. In the context of our self-healing system, goals could be that every component is instantiated somewhere in the system and each component instance is correctly connected.

```
(:action startComponent
  :parameters (?n -Node ?inst - ComponentInstance)
  :vars (?variant - ComponentVariant ?c - Component)
  :condition (and
    (not (instanceWorking ?inst))
    (variants ?c ?variant)
    (instances ?variant ?inst)
    (working ?n)
  )
  :effect (and
    (instanceWorking ?inst)
    (running ?inst ?variant)
    (runningOn ?n ?inst)
  )
)
```

Algorithm 1: action startComponentInstance

We employed the planning software SGPlan in order to compute repair plans for such self-healing systems. Algorithm 1 shows the specification of an action which instantiates a non-working component. This action nicely resembles the story diagram of Figure 2 without the event handling. The event handling is not a part of the planning actions as they are a part of the simulation environment.

Note that the PDDL does not support the creation and deletion of objects. Therefore, we had to simulate that by the predicate instanceWorking.

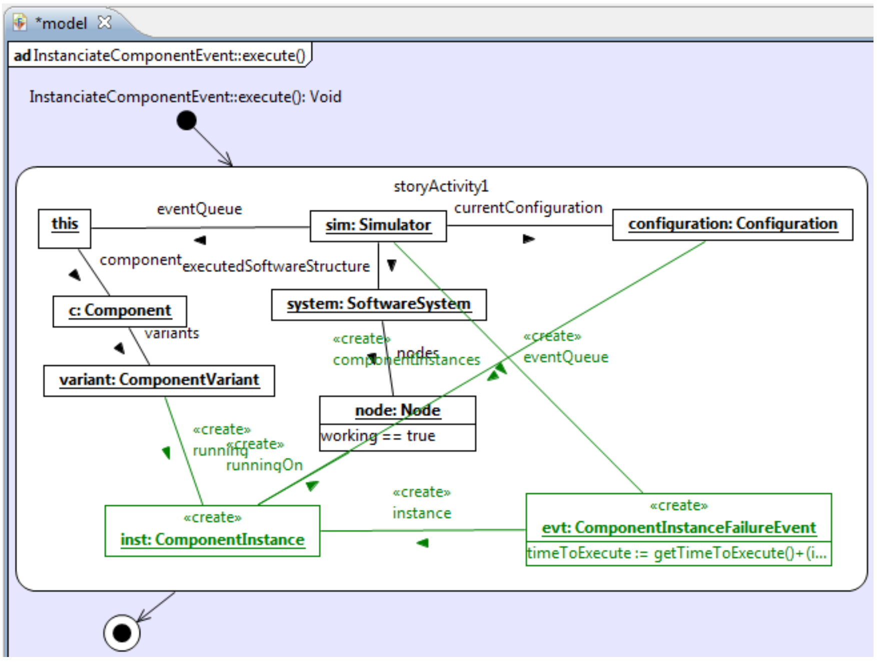

Figure 2: Event for instantiation of a component.

The following is a repair plan which is returned by SGPlan which not only instantiates component CI1 but also moves other component instances to other nodes (e.g. CI2 from node E2 to node E4) in order to free up resources for the instantiation of CI1 on node E2.

```
0.001: (UNCONNECTCOMPONENTINSTANCESBUS CI2 CI3 B2) [1]
1.002: (STOPCOMPONENT E2 CI2) [1]
2.003: (STARTCOMPONENT E2 CI1) [1]
3.004: (STOPCOMPONENT E4 CI3) [1]
4.005: (STARTCOMPONENT E3 CI3) [1]
5.006: (STARTCOMPONENT E3 CI2) [1]
6.007: (CONNECTCOMPONENTINSTANCESVIABUS CI1 CI2 B1) [1]
7.008: (CONNECTCOMPONENTINSTANCESVIAECU CI2 CI3) [1]
```

3.5 Integration with automated planners

Finally, the Fujaba models are integrated with the plans which are returned by the SGPlan. For this, the Fujaba models had to be closely aligned to the PDDL specification. Then, after execution of the planner the resulting plan can be executed by calling the story diagrams with the correct arguments.

Due to time reasons, we did not consider the self-healing scenario in these practical assignments., but only the simpler ferryman problem.

4. CONCLUSIONS

We presented a course on modeling self-adaptive systems with a focus on graph transformations in this paper. The course is built around the three layer architecture for self-adaptive systems and introduces techniques for several parts of this architecture. We did focus on graph transformations as formalism for the specification of actions. The course was accompanied by practical assignments which heavily employed existing tools like Fujaba.

4.1 Lessons Learned

We and the students learned a lot about self-adaptive systems as well as the pro and cons of requiring working with concrete software tools. Overall the practical work with tools increases the motivation of the students as they do not only write text and draw boxes and lines on paper but will test, refine, and improve their solutions until they work. This is difficult using only pen and paper. However, the employed tools have to be more polished than the typical research prototype.

Concerning Fujaba, we decided to give the students an Augsburg version of Fujaba4Eclipse which we also used to develop our sample solutions. According to our knowledge, in Paderborn, Kassel, Tartu and elsewhere different individual versions of Fujaba4Eclipse are given to the students, too. We did make an Ubuntu virtual machine available which contained this Fujaba version as well as GROOVE and SGPlan.

In our opinion, it would be beneficial to join forces and develop and maintain a single version of Fujaba for Education similar to the old Fujaba Life [15] which would be available at a central location.

This Fujaba for Education needs better, central and up-to-date documentation than we have today. We provided screencasts to demonstrate the standard activities like modeling of class and story diagrams as well as code generation and the integration with eDOBS similar to screencasts in Kassel and Tartu. A central location for those screencasts in the documentation (which must align with a single educational version of Fujaba) may also help.

Concerning Fujaba, the students were impressed with the idea of graphical modeling of graph transformations and subsequent code generation that can be actually used in normal

Java programs

Besides the question for better documentation, the students did ask for automatic or better layouting. Another topic was the support of design level debugging [6] in our version of Fujaba which would greatly help in testing and debugging the modeled specification. Finally, a better support for error messages especially during code generation was on the students' wish list. The error messages (e.g. by the sequencer) should be more specific about the error's location or even feed errors back as annotations or markups in the diagram.

We currently work on an automatic generation of PDDL specifications from Fujaba models similar to [2]. PDDL actions more or less resemble story patterns but are more powerful since universal and existential quantifiers are supported in single actions. This was heavily used by the students when implementing the PDDL actions. For a better integration, we lobby for the inclusion of universal and existential quantifiers in the new common SDM-Ecore model as proposed by Stallmann [16] which is similar to the GROOVE syntax.

5. ACKNOWLEDGMENTS

Matthias Tichy is currently on leave from the Software Engineering Group at the University of Paderborn.

6. REFERENCES

[1] B. Becker, H. Giese, S. Neumann, M. Schenck, and A. Treffer. Model-based extension of autosar for architectural online reconfiguration. In S. Ghosh, editor, *MoDELS Workshops*, volume 6002 of *Lecture Notes in Computer Science*, pages 83–97. Springer, 2009.

[2] S. Edelkamp and A. Rensink. Graph transformation and ai planning. In S. Edelkamp and J. Frank, editors, *Knowledge Engineering Competition (ICKEPS)*, Canberra, Australia, September 2007. Australian National University.

[3] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language. In G. Engels and G.Rozenberg, editors, *Proc. of the* 6^{th} *International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany*, LNCS 1764. Springer Verlag, 1998.

[4] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[5] E. Gat. Three-layer architectures. *Artificial Intelligence and Mobile Robots*, 1997.

[6] L. Geiger and A. Zündorf. Design level debugging with fujaba. In *International Workshop on Graph-Based Tools (GraBaTs), Barcelona, Spain*, 2002.

[7] M. Ghallab, D. S. Nau, and P. Traverso. *Automated Planning: Theory and Practice.* Morgan Kaufmann, May 2004.

[8] M. H. Kacem, A. H. Kacem, M. Jmaiel, and K. Drira. Describing dynamic software architectures using an extended uml model. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1245–1249, New York, NY, USA, 2006. ACM Press.

[9] B. Klöpper, J. Meyer, M. Tichy, and S. Honiden. Planning with utilities and state trajectories constraints for self-healing in automotive systems. In *Proc. of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Budapest, Hungary, September 27-October 1, 2010*, 2010.

[10] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering*, pages 259–268. IEEE Computer Society, Washington, DC, USA, 2007.

[11] D. L. Métayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, 1998.

[12] F. Nafz, H. Seebach, J. Holtmann, J. Meyer, M. Tichy, W. Reif, and W. Schäfer. Designing self-healing in automotive systems. In *Proc. of the 7th International Conference on Autonomic and Trusted Computing (ATC 2010), Xi'an, China, 26-29 October, 2010.* Springer Verlag, 2010.

[13] M. Parashar, editor. *Autonomic computing: concepts, infrastructure, and applications.* CRC Press, 2007.

[14] A. Rensink. The GROOVE simulator: A tool for state space generation. In J. Pfalz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *Lecture Notes in Computer Science*, pages 479–485. Springer Verlag, 2004.

[15] C. Schulte, J. Magenheim, J. Niere, and W. Schäfer. Thinking in objects and their collaboration: Introducing object-oriented technology. *Computer Science Education*, 13(4), December 2003.

[16] F. Stallmann. *A Model-Driven Approach to Multi-Agent System Design.* PhD thesis, University of Paderborn, Germany, 2009.

[17] G. Taentzer, M. Goedicke, and T. Meyer. Dynamic change management by distributed graph transformation: Towards configurable distributed systems. In *TAGT'98: Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, pages 179–193, London, UK, 2000. Springer-Verlag.

[18] M. Tichy, S. Henkler, J. Holtmann, and S. Oberthür. Component story diagrams: A transformation language for component structures in mechatronic systems. In *Postproc. of the 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems (OMER 4), Paderborn, Germany.* HNI Verlagsschriftenreihe, 2008.

[19] M. Wermelinger and J. L. Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Sci. Comput. Program.*, 44(2):133–155, 2002.

[20] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *RE 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31 - September 4, 2009*, pages 79–88. IEEE Computer Society, 2009.