The SceBaSy PlugIn for the Scenario-Based Synthesis of Real-Time Coordination Patterns for Mechatronic UML*

Holger Giese and Sergej Tissen
Software Engineering Group, University of Paderborn,
Warburger Str. 100, Paderborn, Germany
[hg|serti]@uni-paderborn.de

ABSTRACT

The future generation of networked, technical applications demands support for the development of high quality software for the proper real-time coordination of safety-critical systems. In this paper, we present the SceBaSy plugin for the Fujaba Real-Time Tool Suite which supports the scenario-based synthesis of the real-time coordination patterns. Extending our approach for the compositional formal verification of MECHATRONIC UML models described by components and patterns [5], the plugin enables the designer to specify the required real-time coordination using multiple parameterized scenarios described by a subset of the UML 2.0 sequence diagram notation. In addition to the synthesis, comfortable analysis capabilities have been realized to guide the designer when conflicts between the different scenarios exist.

1. INTRODUCTION

In the development of safety-critical systems, the design and verification of the real-time coordination of the system is of crucial importance. The increasing complexity of these systems and their interconnection by networks which can be often observed today, makes their production an even greater challenge. Therefore, the current practice could benefit from more automated support for the design of correct and safe real-time coordination.

Today, a number of scenarios are usually developed in the earlier phases to outline and specify possible or required interaction behavior. Later, an operational model of this interaction is then derived manually. The underlying idea of scenario-based synthesis is simply to automate this step (cf. [7, 8, 12, 13]).

However, in our specific case of real-time systems, besides the causal relation between the different events also the timing constraints are essential. Available approaches cf. [10, 9, 6] only provide a global behavior for fixed timing constraints. We employ here our approach [4] for the synthesis of distributed operational behavior from parameterized scenarios, as it is in practice often difficult to specify all timing information such as worst-case execution times (wcet), deadlines, or timeouts in advance.

In this paper, the support for the scenario-based synthesis of real-time coordination pattern provided by the SceBaSy plugin is presented. It complements our MECHATRONIC UML approach for the compositional formal verification [5] which supports to build a correct and safe real-time coordination of a whole system composed of components and patterns.

The plugin supports the automatic derivation of the parameterized role behavior in form of Real-Time Statecharts (RTSC) [2] for patterns from a given set of parameterized scenarios. Therefore, it enables the designer to automate this otherwise costly development step and guarantees correctness by construction. In addition to the synthesis, comfortable analysis capabilities have been realized to guide the designer when conflicts between the different scenarios exist.

The paper is organized as follows: In Section 2, the real-time modeling with scenarios as supported by the SceBaSy plugin is sketched introducing a simple example pattern. Then, the support of the plugin for the analysis of a set of scenarios in case of conflicts is outlined in Section 3. For a conflict free set of scenarios, the handling and output of the synthesis step are then described in Section 4. Finally, we sketch the architecture of the plugin and its dependencies w.r.t. other MECHATRONIC UML plugins in Section 5 and sum up the paper with a short conclusion.

2. REAL-TIME MODELING

As a running example, we consider the *Monitor-Actuator Pattern* [3]. This pattern specifies a controller which monitors and controls another system. Therefore, the controller sends advices to the system via an actuator and monitors their realization. The actuator calculates the actions which have to be done to realize the system-state and sends them to the system. The monitor waits for the system status and decides then whether the advice is fulfilled or not. Furthermore the monitor and actuator check their presence by sending periodically a life tick message to each other (cf. heart beat).

To describe the variable behaviors of the roles within the pattern, we model the different scenarios by UML sequence diagrams where besides constant timing constraints also parameterized timing constraints for upper bounds are supported. The sequence diagrams are modeled with the Fujaba plugin UMLSequenceDiagrams. Additionally, our plugin extends the sequence diagram plugin by time observations and time restrictions to be able to specify real-time behavior within sequence diagrams. Time observations assign the actual time to a clock and time constraints can refer to these clocks and make restrictions. Two sequence diagrams which describe two scenarios of the *Monitor-Actuator Pattern* are shown in Figure 1 and 2

*This work was developed in the course of the Special Research Initiative 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

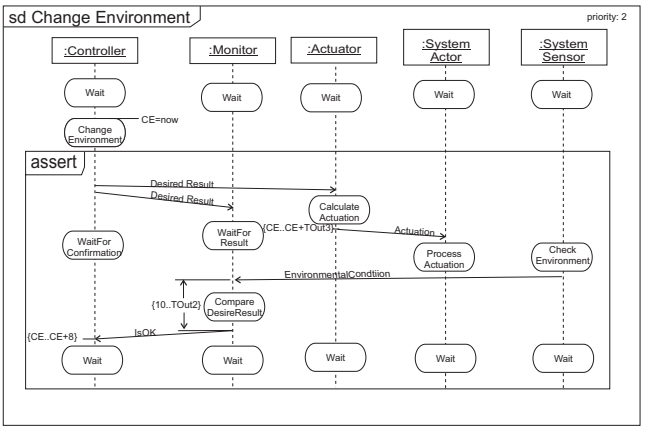

Figure 1: Change Environment

Figure 1 shows the scenario which models the standard system flow. The controller initiates a state change and awaits the result. The controller sends a message to the actuator who then calculates the necessary actions and sends them to the system. Also the monitor receives the same message from the controller to observe the achievement. After receiving the system-state, the monitor compares it with the desired state and sends an okay message to the controller. After at most $TOut_3$ milliseconds (ms), the actuator has to complete his calculations and send his advices to the system. The maximal available ms for the monitor are $TOut_2$. The whole activity must not exceed 8 ms.

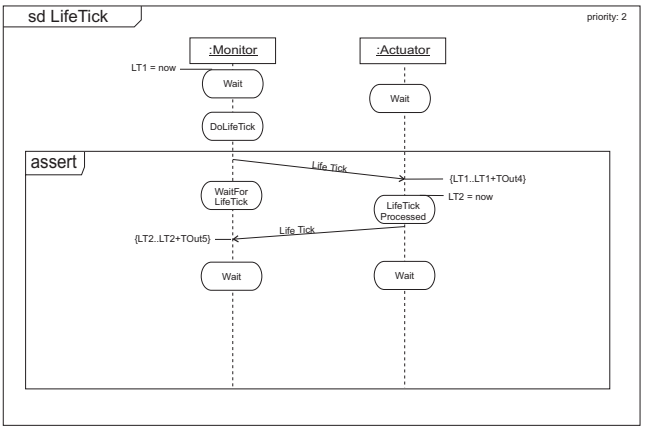

Figure 2: Life Tick

Figure 2 specifies a scenario where the monitor sends a life tick message to the actuator and the actuator responds on his part with a life tick. The monitor has to send a life tick every $TOut_4$ ms and after receiving the life tick the actuator has maximal $TOut_5$ ms to answer.

After we have modeled the scenarios by sequence diagrams, we must create a new synthesis task. When we have entered a name for the task, a new tab named *Synthesis* is generated in the project tree. This panel manages all created scenario-based synthesis-tasks. Figure 3 displays the user interface for synthesis tasks. The root node is the name of the synthesis. The subnode *Sequence Diagrams* holds and manages all sequence diagrams which were imported into the task. The *Settings* node allows setting, removing or modifying additional inequalities which restrict the parameters employed in the sequence diagrams. In addition, we can set weight for all parameters to define which ones should be preferred in contrast to others. The subnode *Pattern* displays the synthesized real-time statecharts and their properties.

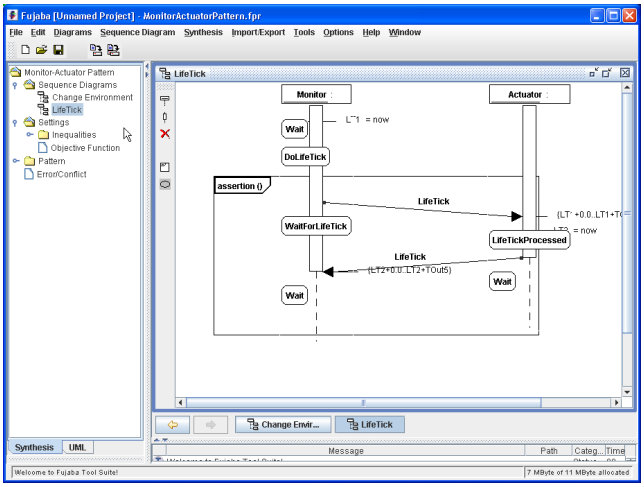

Figure 3: SceBaSy User Interface

3. ANALYSIS

To analyze the sequence diagrams and their timing constraints, our plugin first maps the sequence diagrams to time constraint graphs (TCGs). A TCG represents all possible paths within a sequence diagram and formalizes time observations and constraints. TCG nodes depict possible states of the roles within a sequence diagram, and edges are used to describe how time passes on the lifeline.

In sequence diagrams, *activities* are used to describe the execution of a side-effect. We assume that the specific execution time of an activity is usually unpredictable. However, we can assume lower and upper bounds (cf. worst-case execution times (wcet)) for the activity. The communication in sequence diagrams can be asynchronous or synchronous. Unlike the asynchronous communication in sequence diagrams, TCGs only provide synchronous communication. To address this problem, our plugin generates additional channels which simulate asynchronous communication via buffering.

The plugin maps the timing constraints used in sequence diagrams to constraint edges with a uniquely determined starting point, when the time observation is set, and an end point denoted by the time constraint itself. To reflect the assert blocks within the sequence diagrams, the plugin represents nodes which relate to a state within an assert block in the sequence diagram by assert nodes, all other nodes become possible nodes.

To verify the correct timing behavior of the TCGs, we have to take *consistency* into account which requires that always the same time will elapse on two alternative paths between two nodes. In addition, we demand *locality* [4], which requires that the timing of local tasks only depends on the current state.

To address the problem of *consistency* and *locality*, our plugin derives a set of inequalities which describe the execu-

tion time dependencies [4] and checks their feasibility. This includes all subgraphs of the TCGs which result from time observations and time constraints. Please Refer to [4] for a more precise description. A set of linear inequalities is only feasible iff an assignment for all variables exists which fulfills all inequalities.

If the linear inequality system is not feasible, the integrated conflict handler has to find out which inequalities exclude each other [11]. The conflict handler traces back the inequalities to the affected time constraints or manually set restrictions, and shows an error message to the user. In addition, the constraints in conflict are highlighted by the plugin in the related diagrams.

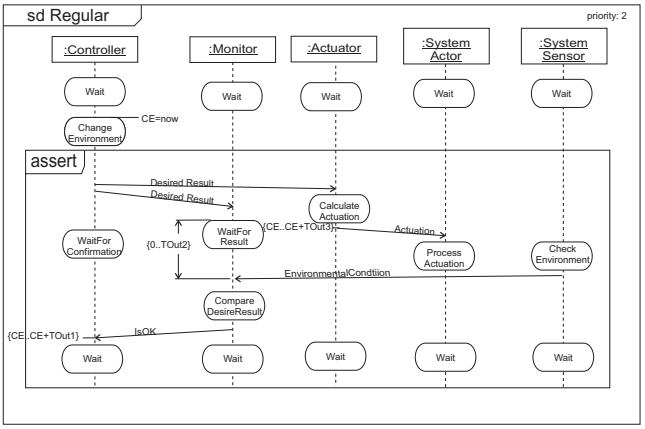

Figure 4: Time Constraint conflict resolved

Figure 4 shows the corrected *Change Environment* scenario where two constraints have been in conflict. The Monitor required min. 10 ms to compare the desired result with the actual environment, in contrast the Controller awaited the result of the calculation within 8 ms (cf. {10 ..TOut2} and {CE ..CE+8} in Figure 1).

In addition to the outlined analysis, asserted blocks have to be taken into account by the inequality system, and we have to check for contradictory state changes in a single state. Please refer to [11] for more details about these and other analysis steps.

4. SYNTHESIS

The previous chapter suggested how sequence diagrams can be analyzed and verified. Now we want to synthesize the behavior for the involved roles. The synthesis-algorithm handles every sequence diagram consecutively. Thus, existent real-time statecharts are extended to the content of a sequence diagram iteratively. One iteration step is as follows: The mapping to states is simply derived for each node using the state labeling of the sequence diagram. Local state changes in sequence diagrams are mapped to transitions between states in a real-time statechart. The communication expressed by a message in a sequence diagram becomes a communication transition in both related statecharts. For the generated transitions, the time conditions simply result in a time guard and deadline such that the specified timing constraints are satisfiable. For more details about mapping sequence diagrams to real-time statecharts refer to [4].

After generating operational behavior for each role, the SceBaSy plugin is able to optimize the real-time statecharts [4]. The optimization algorithm is able to collapse redundant states and their transitions which result from the synthesis into a single state as well as hierarchical states. This optimization can be turned on or off for every pattern role individually. The application of these syntactical rules erases 9 states from the monitor's real-time statechart and results in the model depicted in Figure 5.

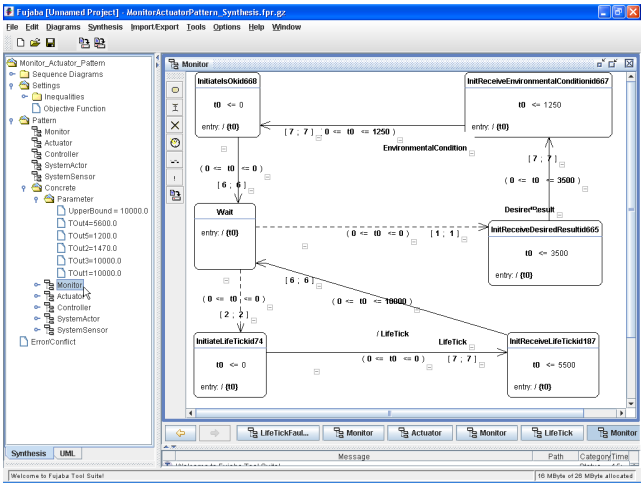

Figure 5: Optimized behavior for the Monitor

Once valid parameters are calculated and real-time statecharts are derived, the plugin uses the model checking capability of the real-time version of Fujaba to ensure that the synthesis result for the given parameter values is free from deadlocks or time stopping deadlocks.[1] After the pattern is successfully model checked, the entire Real-Time Coordination-Pattern is generated as shown in Figure 6. Then each Role gets its own behavior assigned in the form of a real-time statechart.

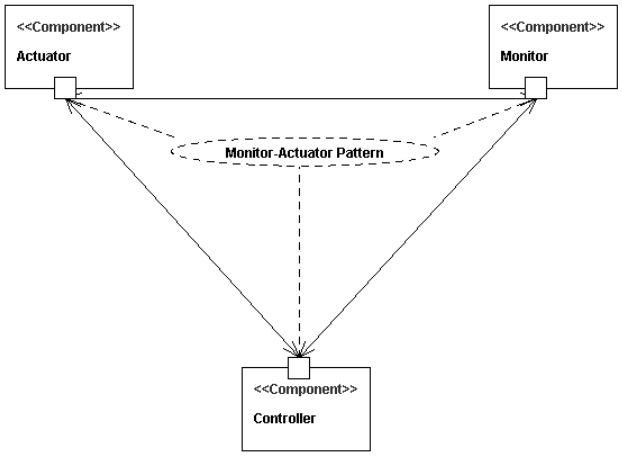

Figure 6: Monitor-Actuator Pattern Structure

[1]It is to be noted, that problems due to time stopping deadlocks and thus reachability can only be proven using a real-time model checker after all parameters have been set, as the emptiness problem for parameterized timed automata with more than 2 parameters is undecidable [1].

The user has now the ability to evaluate the real-time statecharts, to modify the parameter weights, to add or remove inequalities and adapt the sequence diagrams. Now he can initiate the synthesis again to get the readjusted results.

5. PLUGIN

Sequence diagrams used for modeling the scenarios are implemented in a Fujaba Plugin named UMLSequenceDiagrams and the synthesized real-time Statecharts are realized in the plugin RealtimeStatechart. Our plugin extends the UMLSequenceDiagram plugin by the ability of adding timing constraints to sequence diagrams. The plugin UMLRT2 provides the ability to save and restore our generated coordination pattern in a repository.

The SceBaSy plugin provides a standardized interface to solve the linear inequalities. Thus, different inequality solver can be used by the plugin. So far we have made use of two solvers: A java implementation of the simplex algorithm and a commercial package named CPLEX[2].

To evaluate our plugin, we use an extended version of our running example, where the additional sequence diagrams use partially the same parameters in their constraints like the ones presented. We simply add a sequence diagram to the synthesis in every new experiment and record the times and characteristics. Table 1 depicts the results of these experiments. While the simplex package shows a moderate increase in computation time, the commercial CPLEX package does not show an increase in computation time at all. Even though much more experience with the plugin is require to really judge the scalability problem, the experiments are promising.

Number of sequence diagrams	1	2	3	4	5
Numb. of inequalities	54	168	194	220	237
Time in ms to solve the ineq. system (CPLEX)	7	18	21	10	10
Time in ms to solve the ineq. system (Simplex)	19	31	61	69	96
Total runtime [ms]	1170	1930	2056	2671	3102
Numb. of states	13	43	49	55	60
Numb. of transitions	12	44	61	86	95

Table 1: Evaluation data for the SceBaSy plugin

6. CONCLUSIONS

We describe in this paper the SceBaSy plugin for the automatic synthesis of correct real-time coordination patterns from parameterized scenarios. We outlined how the static analysis capabilities of the plugin can be used to analyze problems within a given set of parameterized scenarios. In a next step, the plugin permits to synthesize the real-time behavior for each role of a parameterized real-time pattern in form of parameterized RTSC. In addition, the plugin permits to derive appropriate parameter setting and thus the developer can systematically study the trade-offs between them. When valid parameters for the real-time statecharts have been set, the model checking feature of Fujaba could be used to ensure that the synthesis result is free from deadlocks or time stopping deadlocks.

[2]http://www.ilog.com/products/cplex/

REFERENCES

[1] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 592–601. ACM Press, 1993.

[2] S. Burmester and H. Giese. The Fujaba Real-Time Statechart PlugIn. In *Proc. of the Fujaba Days 2003, Kassel, Germany*, October 2003.

[3] B. Douglas. *Doing Hard Time*. Addison-Wesley, 1999.

[4] H. Giese, F. Klein, and S. Burmester. Pattern synthesis from Multiple Scenarios for Parameterized Real-Timed UML models. In S. Leue and T. Systä, editors, *Scenarios: Models, Algorithms and Tools*, volume 3466 of *Lecture Notes in Computer Science (LNCS)*, pages 193–211. Springer Verlag, April 2005.

[5] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the European Software Engineering Conference (ESEC), Helsinki, Finland*, pages 38–47. ACM Press, September 2003.

[6] D. Harel and R. Marelly. Playing with Time: On the Specification and Execution of Time-Enriched LSCs. In *Proc. 10th IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, Fort Worth, Texas, USA, 2002. (invited paper).

[7] I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to Statecharts. In F. J. Rammig, editor, *Distributed and Parallel Embedded Systems*, pages 61–71. Kluwer Academic Publishers, 1999.

[8] E. Mäkinen and T. Systä. MAS - an interactive synthesizer to support behavioral modeling in UML. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Canada*, pages 15–24, May 2001.

[9] A. Salah, R. Dssouli, and G. Lapalme. Implicit integration of scenarios into a reduced timed automaton. *Information and Software Technology*, 45:715–725, August 2003.

[10] S. Somé, R. Dssouli, and J. Vaucher. From Scenarios to Timed Automata: Building Specifications from Users Requirements. In *Proceedings of the 1995 Asia Pacific Software Engineering Conference (APSEC '95)*, 1995.

[11] S. Tissen. Szenario-basierte Synthese für parametrisierte, zeitbehaftete UML Sequenzdiagramme. Bachelor's thesis, University of Paderborn, Software Engineering Group, Paderborn, Germany, April 2005.

[12] S. Uchitel and J. Kramer. A Workbench for Synthesising Behviour models from Scenarios. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Canada*, pages 188–197, May 2001.

[13] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proceedings of the 22nd international conference on on Software engineering June 4 - 11, 2000, Limerick Ireland*, 2000.