UML-Extensions for Quantitative Analysis*

Konstantinos Kosmidis[1] and Huszerl Gábor[2]

[1] University of Erlangen–Nuremberg, Dept. of Computer Science III
Martensstrasse 3, D–91058 Erlangen, Germany
kk@cs.fau.de
[2] Budapest University of Technology and Economics,
Dept. of Measurement and Information Systems
H-1521 Budapest, Pázmány Péter sétány 1/d., Hungary
huszerl@mit.bme.hu

1 Introduction and Motivation

The Unified Modeling Language (UML) finds more and more applications. It is not only used for software development but also for modeling systems with dynamic behavior (e.g. Flexible Manufacturing Systems (FMS), or Business Process Modeling). While the static diagrams of UML were changed marginal in the latest versions of UML, the dynamic diagrams need still to be improved. For modeling truly the dynamic behavior by these diagrams a concept of time is needed. Therefore, we introduce in this paper an approach to extend UML with time as stochastically variable. To evaluate the enhanced models and to drive a numerical analysis we use the Petri-Net analysis tool PANDA[1]. This tool works also on some stochastic extensions of the Petri-Nets (GSPN's), widely used for performance evaluation and numerical analysis. The evaluation of these models is based on exploring and solving the underlying Markov-chains.

2 An Approach to Extend UML with Time

In this section we give extended semantics on UML-Statecharts suited to stochastic modeling (preserving the choice of possible interpretations given in the UML standard), and present the transformation of the (extended) UML-Statecharts to Stochastic Reward Nets (SRN's).

To be able to use the classical analytical methods and tools for the numerical analysis of UML-Statecharts, we only allow negative exponential distributions for transition times (or approximations with phase type distributions). Because of the limitations of the actually applicable solving tools the size of the total state space of the underlying Markov-chain of the model may not exceed c.a. one million states. Otherwise it is impossible (at the moment) to compute results.

* This work was partially supported by the "Hungarian-German Researchers Exchange Program" (DAAD-MÖB) Project–No. 8, by the "Hungarian Scientific Research Fund" No. OTKA-F030553 and OTKA-T030804 and by the "Hungarian Ministry of Education" Project–No. FKFP-0193/1999

[1] Petri net ANalysis and Design Assistant

To analyze a system it can be modelled by UML-Statecharts with additional labels on the transitions which contains the timing or probabilistic information. In the current implementation the description of this information has an own language, but later it will be made OCL-conform. In several test implementations the transformation of the extended statecharts was executed by add-on scripts of the applied UML CASE-tool, or the UML model was exported to a database and some CASE-tool independent scripts of the database have done the work. After implementing a new graph transformation method [1] a new, more general way of model transformation can be applied.

Based on the Petri-Net implementation of the UML-Model we can compute measures like:

- the expected probability of state configurations of the UML model,
- throughput of transitions of the UML model,
- the expected probability that a transition of the UML model is enabled,
- the expected probability that a transition of the UML model fires,
- (and later) the expected value of some user defined reward functions.

These measures can be defined in the UML model (in the users' world), they can be computed on the transformed model, but because of the systematic way of the transformation, the results can be backannotated and interpreted on the UML model.

Because of the compactness of UML the description of a system with some millions of states results as well in a very large Petri-Net as in a small and clear set of UML-Statecharts. This compactness leads the modeler to make detailed models with large state space. However the modeler has to take into account to build models with amenable state space for today's solving tools.

Another problem caused by introducing explicit time in UML-Statecharts is the step semantic. As long as step semantics (zero-time) in modeling are considered it is possible to say that a step of a UML-Statechart begins with the event dispatching, continues with choosing an appropriate set of triggered transitions, executing actions and state changing, and ends with sending new events. But if all of these "meta-actions" are considered timeless, except of the state change itself, one step begins with waiting for state change, and ends with choosing an appropriate set of triggered transitions. When the stochastically chosen transition times of the fire-able transitions elapsed, all of the other "meta-actions" happen in zero time. Otherwise these model elements of the UML-Statechart would affect the results of the analysis considerably.

In stochastic modeling infrequently firing transitions are customarily modeled by transitions with small transition rates (of negative exponential distribution). Because of the blocking property of the step semantic of the UML, low firing parameters cause long blocking (waiting) times, and the frequency of transition firing is not only determined by its own parameter but also by the parameters of the actually enabled transitions triggered by the same event. The duration of a step is only determined by the longest transition time drawn in the given step. A possible exception represent racing transitions, triggered by the same event, which can be used to model two possible outcomes of a transition (e.g. a

correct one and an erroneous one). In order to avoid that kind of blocking, and to enrich the modeling power of the extended statechart semantic, we allow the use of timer events to trigger transitions. These timers are set (drawn according to the given distribution and parameter) when the explicit source states of the transitions are entered, and they generate events, which trigger the transitions with the given timer.

This way infrequently occurring events can be modeled, and the same construction can be used to model actions with long duration time (for example when modeling hardware components, mechanical processes can have long duration compared to actions of the control software). In this case the action is started by a transition (timeless or with short duration) leading to a state, where another transition with a timer originates. Firing of the second transition models the end of the action.

Furthermore, it is possible to split every timed transition into two separate timeless transitions and a timer. This partition can be done either explicit in the statechart model by additional states representing *"waiting for the timer to expire"*, or during the analysis of the model (e.g. when transforming the model to another platform, such as Petri-Nets or Markov-chains). In the second case, the conformity with the step semantic of the UML (event dispatching and all triggered changes in one single unit) is contradictory.

3 An Example

Our example is a simplified version of the "Trajectory Planner" example of a spacecraft described in [2]. We modeled only the five main objects: Planner, Movement Coordinator, Controller, Sensor and Rocket. The statecharts contain only call- and time-events and no data modeling. A time parameter is associated with each transition, being the parameter of a negative exponential distribution describing the stochastic duration of the transition. For the timers in addition to the transition parameter the timer parameter is given. At the moment the parameters of the transitions are given in the parentheses beside the triggering event.

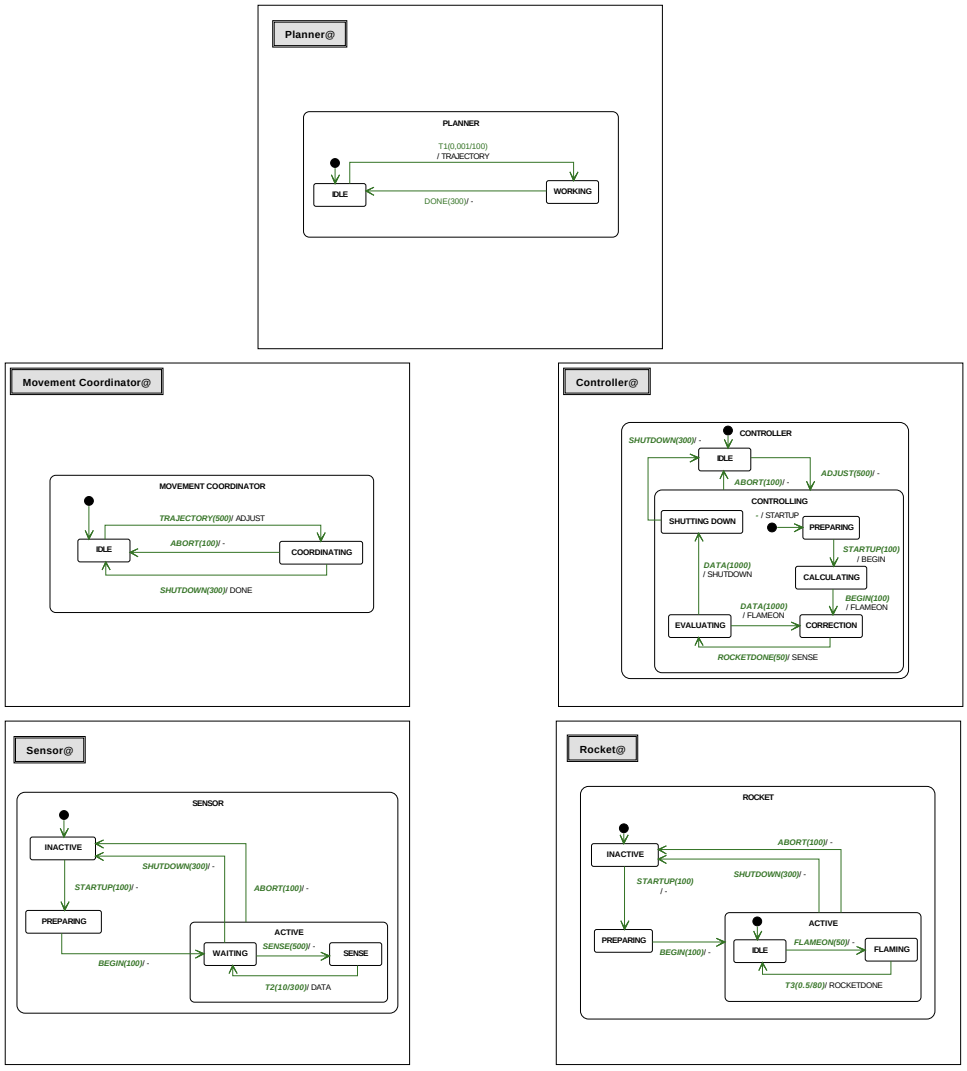
Planner: After calculating a new trajectory, the trajectory can be implemented by the system (moving the spacecraft to the trajectory). When reaching the new trajectory, the Planner continues planning.

Movement Controller: Getting the new trajectory from the Planner the Movement Coordinator triggers the other components with actual data. When the correction of the position of the spacecraft is finished, the Movement Coordinator triggers the Planner.

Controller: Started by the Movement Coordinator, the Controller collects actual data from the Sensor, and triggers the rocket accordingly. It decides, whether additional corrections are necessary to reach the new trajectory.

Sensor: Started by the Movement Coordinator the Sensor sends the actual position to the Movement Controller when asked for.

Rocket: Started by the Movement Coordinator the Rocket (for the sake of simplicity, a single one with controllable nozzles) corrects the trajectory of the spacecraft when asked for.



By the transformation described in [3], [4], [5], and [6] the UML-Statecharts are transformed to a Petri-Net with special consideration of event processing, state-hierarchy and the step semantic of the UML-Statecharts. Due to the simplicity of the example a nondeterministic event dispatcher was chosen. The underlying Markov-chain has 42 tangible states.

Many numerical results of the model can be derived from the the analysis of the underlying Markov-chain. One possible question of the analysis (among the

classical questions of performance evaluation and numerical analysis generally) of this example is: *"What is the expected possibility that the spacecraft completes the correction of the trajectory in a given time?"*. The possibility grows with the time (as expected), passes 90% after 9.81 time units, passes 95% after 12.69 time units.

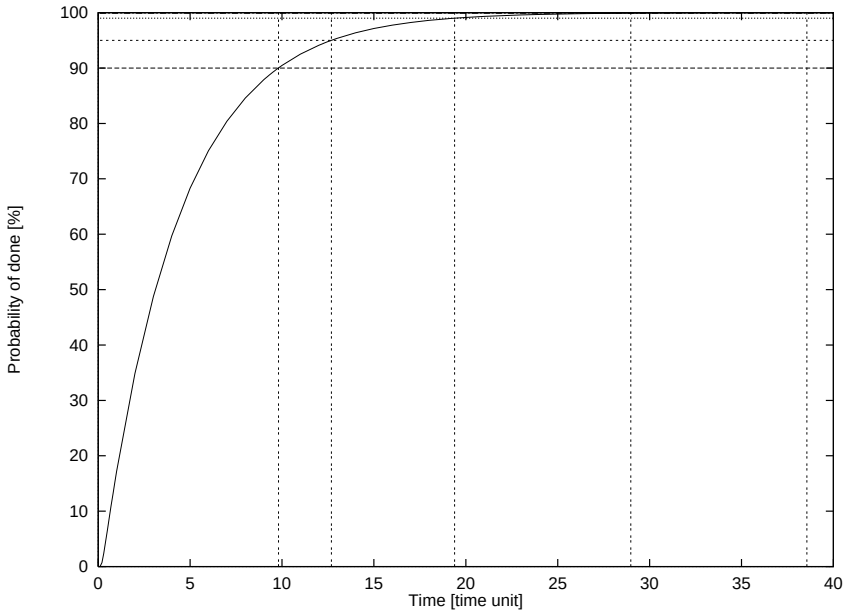


Fig. 1. Numerical results of the example

4 Conclusions

In our paper we presented an approach to combine traditional numerical analysis methods, based on Markov-chains, and the UML. To this end it is necessary to extend the UML-Statecharts with the concept of time as a stochastic variable and to transform them to Petri-Nets. We described some possible time semantics and the effects of them on the step semantics of the UML-statecharts. In order to prevent the problem of the state space explosion, the modeler should take into account to build models with amenable state space. In case that the state space can't be handled by the actual solving tools a simulation can be performed, or scenarios that represent parts of the systems behavior can be evaluated.

References

1. D. Varró, G. Varró and A. Pataricza: *Designing the automatic transformation of visual languages* – In H. Ehring and G. Taentzer, editors, GRATRA 2000 Joint APPLIGRAPH and GETGRATS Workshop on Garph Transformation Systems, pp. 14-21., Technical University of Berlin, Germany, March 2000
2. Bruce Powel Douglass: *Doing Hard Time*, Addison–Wesley, 1999, pp. 427-434
3. M. Dal Cin, Huszerl G., K. Kosmidis: *Transformation of Guarded Statecharts for Quantitative Evaluation of Dependable Embedded Systems* – EWDC-10, Vienna, Austria, 6–7 May 1999, pp. 143-187
4. M. Dal Cin, Huszerl G., K. Kosmidis: *Quantitative Evaluation of Dependability critical Systems Based on Guarded Statechart Models* – In Proc. HASE'99, Washington DC, USA, November 1999, pp. 37-45
5. Huszerl Gábor: *Design Pattern Based Transformation of Dynamic UML-Models for Quantitative Analysis* – EWDC-11, Budapest, Hungary, 11-13 May 1999
6. Huszerl G., Majzik I.: *Quantitative Analysis of Dependability Critical Systems Based on UML-Statecharts-Models* – In Proc. HASE'00, Albuquerque, New Mexico, USA, 15-17 November 2000