DEVELOPING SIMULATION MODELS USING STATECHART METHODOLOGY

Michael Gyimesi[1], Patrick Einzinger[2], Felix Breitenecker[1]

[1] Technical University of Vienna,
1040 Vienna, Wiedner Hauptstraße 8-10, Austria
[2] dwh Simulation Services,
1070 Vienna, Neustiftgasse 57-59, Austria

michael.gyimesi@tuwien.ac.at

Abstract

While object oriented programming became the quasi standard of software development, the object-oriented modelling paradigm has emerged as very useful for modelling of simulation models and object-oriented techniques have been introduced in simulation modelling. Especially statechart diagrams as part of the Unified Modelling Language (UML) have shown to be really useful for discrete event simulation modelling.
David Harel of the Weizmann Institute of Science introduced statechart diagrams in the 1980s. He extended the principles of conventional state-transition-diagram formalism, basically drawing states a system can be in and transitions from one state to another, by the concepts of hierarchy, concurrency and communication.
Thereby statechart modelling became a powerful and at the same time easy to apply method to model arbitrary systems. The first version of statecharts was applicable for just discrete systems, but in the meantime a real time compliant version as part of the UML for real time (UML-RT) - standard, was developed.
While the modelling process is flexible and easy to communicate, the number of simulation software that supports statechart-modelling is increasing. Therefore we introduce statechart methodology as a general method for modelling simulation models and emphasise the usefulness in agent based modelling.

Keywords: Discrete Event Simulation, Hybrid simulation, Statecharts, Object oriented Modelling, UML

Presenting Author's biography

Michael Gyimesi studied technical mathematics at the University of Technology in Vienna and received his PhD in 2005. After that he focused on health service related models and data analysis techniques. His current research interests in modelling and simulation include health services research, object oriented modelling and multi-scale modelling.

1 Introduction

David Harel of the Weizmann Institute of Science introduced the statechart modelling formalism in the 1980s. He extended the conventional state-transition-diagram formalism by the concepts of hierarchy, concurrency and communication [1,2,3]. In the 1990s statecharts were included into the Unified Modeling Language (UML), a set of graphical modelling guidelines and the de facto modelling standard in object oriented programming [4,5].

Meanwhile the object oriented modelling paradigm as method for modelling of systems has emerged as very useful for modelling of simulation models and different simulation software exists that uses object oriented modelling and the enhancements of UML. In particular statechart diagrams and activity diagrams emerged as very useful for simulation modelling [6,7,8].

Therefore we will discuss statechart-modelling methodology from a simulation's point of view. We give a short introduction in principles of statechart modelling in section 2 following in section 3 with some remarks on simulation paradigms and where statecharts can be seen in that context.

In section 4 we give a more realistic example in showing the easy use of system modelling with statecharts and give some concluding remarks in section 5.

2 Principles of statechart modelling

Statechart modelling includes different concepts, namely "regions", "states", "transitions", "pseudostates" and "branches".

- Regions are the areas where states, transitions, pseudostates and connectors are placed in. Regions can be nested hierarchically or can represent concurrent state spaces of an object.
- States are the conditions that an object or a system whose behaviour is described by statecharts can be in and are denoted by rounded rectangles. Inside a state, actions can take place as "entry action" or "exit action". Actions are usually commands respectively programming code depending on the actual modelling environment.
- Transitions define the possible switch from one state to another and are denoted by a unidirected arrow connecting two states. A transition becomes active, when an event inside the system occurs which is connected to the transition's so called guard.
- Pseudostates are statechart objects that are neither states nor transitions and represent special semantics to be applied in a given situation. There are different types of pseudostates:
  - initial state
  - final state
  - shallow history state
  - deep history state
- Branches are statechart objects that connect multiple transition segments. There are two branches: "fork" for outgoing transitions and "join" for incoming transitions.

Figure 1 shows a basic statechart with one region, an initial state (each region needs an initial state as starting point), two states "idle" and "busy" and two transitions to model the possible switches from idle to busy and vice versa. The transition from the initial state to idle is taken immediately.

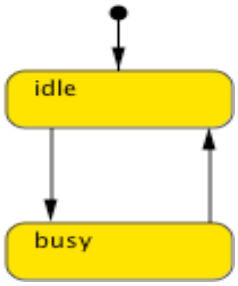

Fig. 1 Basic State diagram

A statechart comprises one or more regions. As the top-level statechart diagram is also a region, there exists at least one region in every statechart. Additionally regions can be created within states, building a nested hierarchy or multiple regions can be placed independently inside one statechart (or one state), thereby building "orthogonal" state spaces. This state spaces act concurrently and change of states inside one region executes independently from state changes in orthogonal regions.

Orthogonal regions can communicate with each other via broadcasting and receiving events. Every region can create an event as a result of a transition that is consumed by another orthogonal region. For every transition inside a receiving region a guard may be used to test if another (sending) region is in a certain state before a transition in the receiving region fires.

Figure 2 shows a small statechart with two orthogonal regions - one region includes the states B and D, the second region just the state C. The moment the statechart becomes active, the system switches in states B and C immediately. The transitions from B to D (and back) respectively from C to a final state are taken independently from each other.

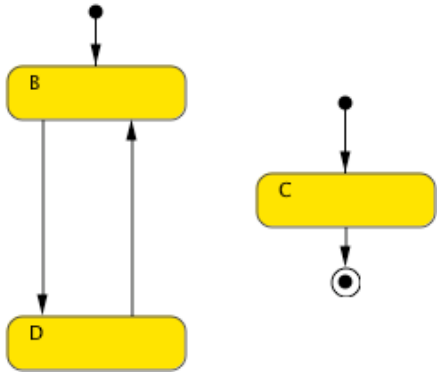

Fig. 2 State diagram with orthogonal regions

The second really important enhancement by Harel was the introduction of hierarchy, represented through nested regions, thereby reducing the rapidly growing complexity of big "flat" models significantly.

Figure 3 shows a statechart with two nested regions. While at top level the statechart starts with immediate transition in state A, the region inside A switches into state B. Depending on the defined transitions or guards, the statechart switches into state C or state D. If state C is active, it can be important that C is not a part of the region inside A.

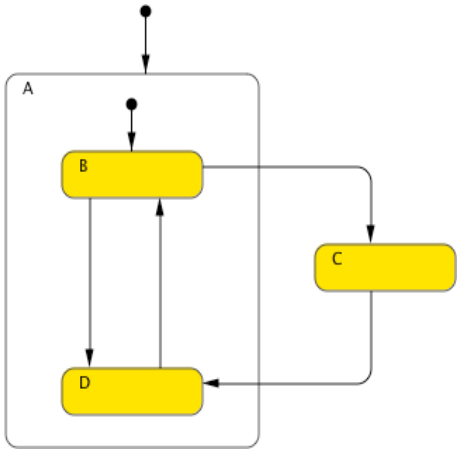

Fig. 3 State diagram with nested regions

3 Statechart Modelling and Simulation

3.1 Statecharts for discrete event simulation

Building models includes always a process of abstraction of reality. This holds for every type of model building including statistical models, physical models and of course simulation models. Which modelling technique somebody chooses depends on the purpose of abstraction respectively the purpose of simulation.

Although there exist basically two more or less distinct application areas in the simulation domain, the discrete event simulation (DES) area and the continuous simulation area, most real systems have a certain amount of discrete behaviour from a simulation's point of view. Therefore most simulation models for real systems use DES or at least some hybrid principles as hybrid means a combination of both DES and continuous simulation (e.g. differential equations).

Inside the DES domain people distinguish between event oriented and process oriented paradigm. Although the mathematical fundamentals of DES come from the event-oriented worldview, most simulation software packages use a process-oriented approach. While events are the moments when the system state changes, processes represent the sum of activities that refer to a specific object. The latter may be a more natural representation of a real system, enabling the modeller to concentrate more on spatial aspects and communicate the model via a graphical representation that includes sources, sinks, processing nodes and connecting edges.

Pure modelling engineers usually neglect the need of communicating their model, but when communication between modeller and specialists in the field of research becomes important, object oriented modelling shows to have a real point. Similar to the emergence of object-oriented programming, the arguably accepted programming standard in recent years, the simulation community adopted the principles of object oriented abstraction and technologies in recent years. One of these useful techniques are statechart diagrams as part of the Unified Modeling Language (UML), which was introduced by Rumbaugh, Booch and Jacobson in the nineties of the last century. They have shown as a powerful modeling approach for discrete event systems.

From the event-oriented viewpoint every activation of a statechart transition schedules an event. States restrict the set of possible events that might be scheduled next, because only the firing conditions of outgoing transitions of the active state (or states in the case of orthogonal regions) of the system are checked. Therefore single events could replace a statechart if there was one event for every transition and the condition for an event incorporated both the condition of the transition it corresponds too and a condition that secures that the event is just scheduled if the statechart was in the right state. However statecharts are preferable because they offer structure that allows modelling the same system in a very intuitive way that is much less error-prone.

3.2 Statecharts for hybrid simulation

The first version of statecharts was applicable just for discrete systems. But in the meantime a modelling standard for real time applications, UML – RT, was developed, that integrates discrete system's behaviour with real time concepts. And as part of this standard, a

newer version of statechart diagrams was developed, which is particularly suitable for hybrid modelling.

An example of a hybrid simulation statecharts are exceptionally suitable for is the “Bouncing Ball” [9]. In models like this the states of the system correspond to different modes of execution. In each of these modes a continuous part (usually a system of differential equations) governs the behaviour of the system. Transitions between the states are often triggered by the state of the differential equation system (for example by zero crossings of one of the state variables) of the active state. For this type of hybrid systems the statechart formalism provides a natural representation.

3.3 Software using statechart methodology

In this section we will introduce some simulation software with statechart technology with no claim to be complete:

- AnyLogic: AnyLogic is multimethod simulation software, developed by XJ Technologies that covers basically three different modelling approaches: system dynamics, discrete event simulation, agent base modelling and any combination of these approaches in one model. That makes it a good choice for all hybrid modelling [10].
- MATLAB Stateflow: Stateflow is a simulation tool for event-driven systems, developed by MathWorks and is integrated with MATLAB and Simulink [11].
- LabView: LabView is a platform and development environment for a visual programming language developed by National Instruments and is platform independent [12].
- MOSILAB: The generic simulation tool MOSILAB is developed by a consortium of different Fraunhofer institutes in Germany and is based on differential algebraic equations [13,14].

Depending on different needs for simulation models, especially integration with other software, one can find some other software packages that use statechart methodology as well.

4 Example for statechart modelling

To explain how to model with statechart diagrams, we introduce the following example. It represents the artificial situation (or “system”), where a working person becomes sick – imagine influenza – and therefore has to go to the general practitioner to get a permit for sick leave, stays at home for duration of the infection and returns to work after he has recovered. There are of course different alternatives to model such a system, but the showcase will show a lot of possibilities the statechart formalism offers (Figure 4).

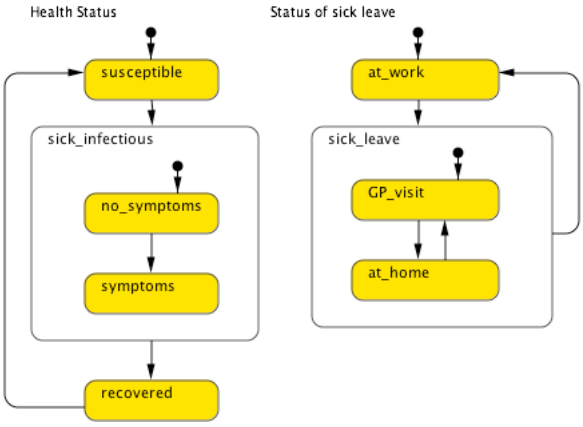

Fig. 4 State diagram with orthogonal regions

The system respectively person at the beginning is healthy but susceptible and working as usual. As the two properties of the person are not necessarily depending on each other, we represent this in a statechart with two independent, orthogonal regions. We will not focus on how the person gets infected but if this happens, the left region switches from state susceptible to state sick_infectious. As we choose an infection with a latency to show symptoms, the individual will be in state no_symptoms first, but at the same time also in state sick_infectous. The right region switches from state at_work to state sick_leave not before the person shows symptoms. A guard for the transition from state at_work to state sick_leave listening to the corresponding switches from state no_symptoms to state symptoms acts as a synchronising mechanism for both regions.

In the showcase health system a person needs a permit for sick leave and has to go to a general practitioner. Consequently the first state inside the state sick_leave is the state GP_visit. After that the person will stay at home as long as the infection endures, showing symptoms of different grade. Let us imagine that the person recovers after a fixed time period. So if that happens in the statechart, a guard in the right region switches from state at_home to state GP_visit, modelling a person's duty to end a sick leave permit. After that the person switches to the state at_work and because the person is immunised for a while, he or she remains in the state recovered for that period and cannot be infected before he or she switches to state susceptible.

As this statechart model is not a unique representation of the system, alternatives are possible or even necessary for an expert in the field. Therefore the model could look like in figure 5.

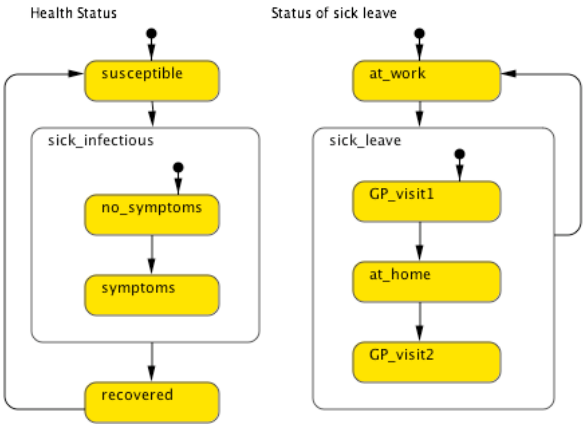

Fig. 5 Alternative example model

The two visits by the general practitioner are decomposed in two different states, which can make sense depending on the purpose of the model.

When we try to model the spread of an infectious disease, we could go for an agent based simulation model, where every agent acts as an individual with the illustrated behaviour. The only additional thing to model the spread of an infection would be the inclusion of an infection mechanism occurring when susceptible and infectious people meet.

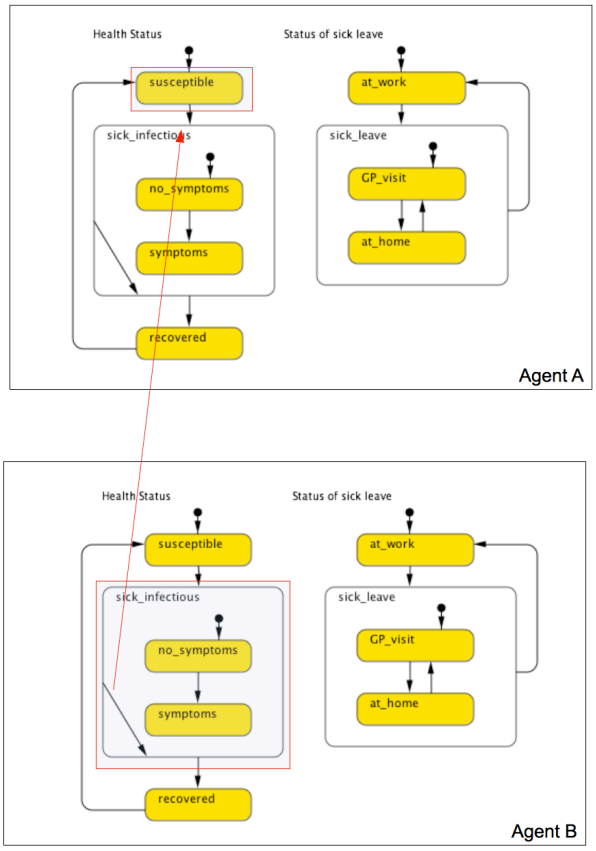

Fig. 6: State diagram with infection mechanism

Just to show how easy this could be represented in a statechart, we show one (of many possible) solution to implement this in figure 6. If agent A and agent B meet with A in state susceptible and B in state sick_infectious, a transition inside B's sick_infectious state (independent, if there are symptoms or not, and without changing B's state) triggers the switch from state susceptible to state sick_infectious for agent A, thereby acting like independent orthogonal regions inside the system of the two agents.

5 Conclusion

Statecharts were introduced as an enhancement of traditional state machine modelling and improved state machines by the addition of concepts for multiple regions, hierarchical states and communication broadcasting. These small additions made statechart modelling extremely powerful and an even newer enhancement introduced statecharts for real time modelling. Thereby statechart diagrams provide a flexible, while not unique but descriptive and easy to communicate way to model systems with only a handful of syntactical rules.

6 References

[1] D. Harel, "Statecharts: A visual Formalism for complex systems", The Science of Computer Programming, 8, pp.231-274. 1985

[2] Harel, D. (1988). On Visual Formalisms. Communications of the ACM 31, 5. 514-530. May 1988

[3] P. Ward and S. Mellor, Structured Development for Real-time Systems, Prentice Hall pp.86 & 302. 1985

[4] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language User Guide, Addison Wesley, 1999.

[5] Object Management Group. Omg unified modeling language specification, version 1.5. http://www.omg.org/cgi-bin/doc?formal/03-03-01. 2003

[6] Coleman, D., F. Hayes, S. Bear (1992). Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design. IEEE Transactions on Software Engineering 18, 1, 9-18. 1992

[7] Rubin, K.S., A. Goldberg (1992). Object Behavior Analysis. Communications of the ACM 35, 2, 48-62. 1992

[8] B.P. Douglass, "UML Statecharts", http://www.embedded.com/1999/9901/9901feat1.htm. 1999

[9] R. Karim, P. Bauer, B. Heinzl, M. Rössler, G. Schneckenreither, A. Körner, K. Breitenecker (2010). Hybrid State Chart Modelling for Nonlinear Bouncing Ball Dynamics. Proceedings Eurosim. 2010

[10] AnyLogic by XJ technologies. http://www.xjtek.com/

[11] Matlab and Stateflow by MathWorks. http://www.mathworks.com.

[12] LabVIEW by National Instruments. http://www.ni.com/labview/statechart.htm

[13] Nytsch-Geusen, C. et. al., ”Advanced modeling and simulation techniques in MOSILAB: A system development case study”, Proceedings of the 5th International Modelica Conference, Arsenal Research Wien, 2006.

[14] "Modelling Structural Dynamic Systems: Standard Modelica vs. Mosilab Statecharts"; "Proceedings MATHMOD 09 Vienna - Full Papers CD Volume". 2009