AN INTRODUCTION TO STATECHARTS MODELLING AND SIMULATION

Simon Van Mierlo
University of Antwerp

Antwerp, Belgium
Simon.VanMierlo@uantwerp.be

Hans Vangheluwe
University of Antwerp
Flanders Make, McGill University
Antwerp, Belgium
Hans.Vangheluwe@uantwerp.be

ABSTRACT

Statecharts is a formalism to model timed, reactive, autonomous systems. It uses a discrete-event abstraction: state changes can occur when an event arrives from the environment, or if one is raised locally. Statecharts are successfully used to model user interfaces, embedded controller software, artificial intelligence, and much more. Many tools have been developed to model, simulate, and generate code from Statecharts. In this tutorial, we introduce Statecharts and explain how they can be modelled, simulated, and debugged using a visual modelling and simulation interface. We start from the basic (non-hierarchical) concepts of states and transitions, and then move on to more advanced concepts of hierarchy, concurrency, and history. We use a simple traffic light application as a running example to demonstrate each concept. Although we focus on the practical aspect of modelling with Statecharts, we introduce the theoretical underpinnings along the way. We use Yakindu, a Statecharts modelling and simulation environment built on top of the Eclipse IDE, but the techniques taught in this tutorial can be applied to any other Statecharts modelling and simulation tools.

Statecharts, introduced by Harel (Harel 1987), is used to specify complex, timed, reactive, autonomous discrete-event systems. It is an extension of Timed Finite State Automata which adds depth, orthogonality, broadcast communication and history. Its visual representation is based on higraphs, which combine graphs and Venn diagrams (Harel 1988). This representation is most suited to represent Statecharts models, and many tools offer visual editing and simulation support for the Statecharts formalism. Examples include STATEMATE (Harel and Naamad 1996), Yakindu (https://www.itemis.com/en/yakindu/statechart-tools/), Rhapsody (Harel and Kugler 2004), and Stateflow (https://www.mathworks.com/products/stateflow.html).

This tutorial introduces Statecharts modelling, simulation, and testing. As a running example, the behavior of a simple timed, autonomous, reactive system is modelled: a traffic light. We start from the basic concepts of states and transitions and explain the more advanced concepts of Statecharts by extending the example incrementally. We use Yakindu to model the example system.

Statecharts is used to model the behaviour of reactive systems. A reactive system receives input from the (external) environment and produces output back to the environment. Since Statecharts uses a discrete-event abstraction, this communication with the environment are abstracted as events. In this tutorial, a traffic light as an example and model its behaviour using Statecharts is presented. This example is basic, but has all essential complexity: it is timed, since its lights have to switch autonomously, and it is reactive, since its normal execution can be interrupted by a policeman. As will be explained in the tutorial, to effectively develop these systems, traditional approaches based on threads and timeouts add accidental complexity, and Statecharts is more suited.

The basic building blocks of Statecharts models are states (modelling the configuration of a system) and transitions (modelling the dynamics of a system, i.e., how the system's configuration changes over time).

Transitions are triggered by an event coming from the environment, or an internal event. It can raise events to the environment. The tutorial starts with modelling a basic autonomous traffic light that only uses these two basic abstractions. Then, the example is extended with three advanced Statecharts concepts: depth (in the form of composite states), orthogonality (in the form of orthogonal components) and history (in the form of history states). Composite states can nest states (arbitrarily deep) in order to group common behaviour. Orthogonal states introduce a notion of orthogonally executing regions that can communicate using (internal) events. History states allow to restore the last configuration of a composite state when it is re-entered. During the tutorial, the example system will be progressively updated and extended with these advanced concepts. Simultaneously, the semantics of these constructs will be explained.

To successfully develop systems using Statecharts, techniques for testing whether the model(s) meet(s) the system's requirements are necessary. To do this, in the tutorial, a test scenario, which consists of a series of input events, and its expected output trace, will be selected. The tested model is subsequently placed in an "experimental frame" (Zeigler, Praehofer, and Kim 2000) to check whether the test case passes. To model the execution of this test case, it is modelled in Statecharts as well: the input and output models are coupled to the tested model to simulate the interaction with the environment.

ACKNOWLEDGEMENTS

This work was partly funded by a PhD fellowship from the Agency for Innovation by Science and Technology in Flanders (IWT) and by Flanders Make vzw, the strategic research centre for the manufacturing industry.

REFERENCES

Harel, D. 1987, June. "Statecharts: a Visual Formalism for Complex Systems". Science of Computer Programming vol. 8 (3), pp. 231–274.

Harel, D. 1988, May. "On Visual Formalisms". Commun. ACM vol. 31 (5), pp. 514–530.

Harel, D., and H. Kugler. 2004. Integration of Software Specification Techniques for Applications in Engineering: Priority Program SoftSpez of the German Research Foundation (DFG), Final Report, Chapter The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML), pp. pp. 325–354. Berlin, Heidelberg, Springer Berlin Heidelberg.

Harel, D., and A. Naamad. 1996, oct. "The STATEMATE Semantics of Statecharts". ACM Trans. Softw. Eng. Methodol. vol. 5 (4), pp. 293–333.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. Theory of Modeling and Simulation. second ed. Academic Press.

AUTHOR BIOGRAPHIES

SIMON VAN MIERLO is a PhD student in the department of Mathematics and Computer Science at the University of Antwerp (Belgium). For his PhD, he is studying how modelling formalisms, environments, and simulators can be enhanced with debugging support.

HANS VANGHELUWE is a Professor in the department of Mathematics and Computer Science at the University of Antwerp (Belgium) and an Adjunct Professor in the School of Computer Science at McGill University (Canada). He heads the Modelling, Simulation and Design (MSDL) research lab. He has a longstanding interest in the DEVS formalism and is a contributor to the DEVS community of fundamental and technical research results.