Some Thoughts on Statecharts, 13 Years Later

(Summary of Talk)

David Harel*

Abstract: The talk describes the background and motivation for the language of statecharts, discusses some of the semantic issues it raises, shows how statecharts have been embedded in structured-analysis and object-oriented frameworks, and describes the supporting tools, STATEMATE and RHAPSODY, from i-Logix, Inc. Some peripheral research is also mentioned.

Introduction

Statecharts were conceived of in late 1983. In the (belatedly published) paper that first presented them [H1], statecharts were portrayed in isolation, as a visual formalism for specifying 'raw' reactive behavior. They were described rather informally there, and several semantic issues were discussed only very briefly.

In the years that have elapsed since, much work has been carried out around statecharts. In this talk we attempt to survey some of the main lines of this work, which can be divided up as follows:

1. Semantics of statecharts.
2. Embedding in a structured analysis framework.
3. The STATEMATE tool.
4. Embedding in an object-oriented framework.
5. The RHAPSODY tool.
6. Variants of the language.
7. Related research topics.

Item 3 essentially implements the ideas in item 2, and, similarly, item 5 implements the ideas in item 4. In this note we shall say a few words about items 1, 4 and 7.

Semantics

A rigorous semantics was first defined for statecharts in [HPSS]. Since then, many variants of statecharts have been proposed in the literature, and several papers include definitions of semantics too. Some examples are [HR, KP, L^+, M, PS]. A recent survey [vB] discusses around 20 variants. Subtle issues arise when

* The Weizmann Institute of Science, Rehovot, Israel, and i-Logix, Inc., Andover, MA 01810. Email: harel@wisdom.weizmann.ac.il.

one tries to define a semantics for this language, and there is no consensus on the "right" way to go about the task.

In a recent paper [HN], we describe the semantics of statecharts as implemented in the STATEMATE system [H+, HP]. The initial version of this semantics was developed by several people about 10 years ago. With the added experience of the users of the system it has since been extended and modified. This executable semantics has been in operation in driving the simulation, dynamic tests and code generation tools of STATEMATE since 1987, and a technical report describing it was written in 1989. We decided to revise and publish the report so as to make it more widely accessible, to alleviate some of the confusion about the "official" semantics of the language, and to counter a number of incorrect comments made in the literature about the way statecharts have been implemented.[2]

Being an unofficial language, statecharts clearly has no official semantics, and researchers are free to propose semantics as they see fit. However, for better or worse, the only implemented and working semantics for statecharts has for many years been the one described in [HN]. One of the main issues in defining a semantics for statecharts, and one that was the central topic of our often heated deliberations when working on the problem, is whether changes that occur in a given step (such as generated events or updates to the values of data items) should take effect in the current step or in the next one. The semantics we finally adopted, in contrast to those of [HPSS, PS], for example, takes the latter approach.

The main consideration behind the definitions in [HN] is to provide a semantics for the specification and design of real-world complex systems, coming from a variety of disciplines. As such, the semantics has to be rich enough to support different styles of modeling, yet it should be relatively simple and intuitive. It must also be technically straightforward enough to enable fast simulation of models, and to generate useful hardware and software code out of these models. Clearly, these desires cannot be satisfied in full, but they served to guide the process that led to [HN].

Since the publication of [HN] a number of papers have been written, in which more formal descriptions of this semantics have been proposed. They include [EGKP, MLPS, PU].

Object modeling with statecharts

For large systems, statecharts cannot be used as the sole specification technique, even if behavior is the central concern. Rather, one should use statecharts as the

[2] For example, the survey [vB] does not mention the STATEMATE implementation of statecharts or the semantics adopted for it at all, although this semantics is different from the ones surveyed therein (and was developed earlier than all of them except for [HPSS]).

behavioral component of a general system-modeling approach. This is a nontrivial matter, since the links between the various aspects of a system's description can be subtle and slippery, especially if full behavior is to be part of the specification. Since modeling approaches must be detailed and precise enough to enable model execution, dynamic analysis and code synthesis, the language set must be rigorously defined and 'closed up': Any possible combination of graphical and/or textual constructs must be clearly characterized as syntactically legal or illegal, and all legal combinations must then be given unique and formal meaning.

About a decade ago, a language set was built around statecharts, based on the function-oriented structured-analysis paradigm (SA). Statecharts, used for behavioral description, were closely integrated with a structured language for functional decomposition and data-flow, called activity-charts. A third language, module-charts, was used to specify physical decomposition. See [H3] for the motivation and 'philosophical' aspects of this effort, and [HP] for a full description of the languages. This language set underlies the STATEMATE tool, built to enable executability, analysis and full code-generation [H+]. However, since SA methods are widely regarded as suffering from a discontinuity problem in transition to design and reuse, many people recommend approaches that follow the object-oriented paradigm. This change is one of the most significant in software engineering in recent years. Accordingly, we recently embarked on an effort to develop a set of languages for object modeling, built around statecharts, and to construct a supporting tool (called RHAPSODY) with full executability and code-synthesis capabilities. This effort is reported on in [HG].

The basic idea is to model the structural properties of classes in a hierarchical manner, and to integrate the resulting description with a precise specification of behavior over time, using statecharts. Since classes represent collections of concrete objects (instances), and since the instances and their relationships change dynamically over time, the model must address issues like the initialization of, and the reference to, actual object instances, the delegation of messages, the creation and destruction of instances, the initialization, modification and maintenance of links representing association relationships, etc. We must also define aggregation and inheritance from a behavioral point of view. All this makes the problem of combining structure and behavior much harder than in an SA-based framework. And it is particularly delicate in the realm of reactive systems, which are characterized not by data-intensive computation but by control-intensive, often time-critical, behavior [HPn].

Most object-oriented modeling methodologies (for example [B, CD, R+, SGW, SM]) offer graphical notations for specifying the model. They typically have ER-style diagrams for specifying classes of objects and their inter-relationships, and means for describing the interface and capabilities of the objects themselves. A state-based formalism is usually adopted for specifying behavior, and the methodologies listed above recommend statecharts (or some sublanguage thereof) for this. However, in many cases such methodologies do not address dynamic semantics adequately, so that the precise behavior of models over time

is not always well-defined. One major motivation for our work in [HG] was to eliminate this crucial shortcoming.

The approach in [HG] involves two constructive modeling languages, *object-model diagrams* and *statecharts*, and a reflective language, *message sequence charts* (MSC's; also called sequence diagrams).[3] Object-model diagrams specify the structure of the system, by identifying classes of objects (i.e., object types) and their multiplicities, object relationships and roles, subtyping and inheritance. Especially noteworthy in this language is the provision for specifying composite objects, which capture a strong form of aggregation; they are depicted by higraph-like encapsulation [H2].[4]

A statechart attached to a class specifies the behavior of all instances thereof. It captures not only the state of the object in terms of its willingness, as a server, to respond to events or requests for services, but also the dynamics of its internal behavior in carrying out those responses, and in maintaining its relationships as a client (or aggregate) with other objects.

One of the main technical issues that arise in devising the setup concerns the choice of mechanisms to be used for inter-object interaction. There is a tradeoff between high-level mechanisms that are easier to model, and lower-level ones that are easier to translate into efficient code. We have tried to compromise in [HG], adopting a two-kind approach — asynchronous *events* and synchronous *operations* — and in the process make careful distinctions between them. An object can generate an event, which is queued, to be "plucked" from the queue by the target server object in its turn, and an object can also directly invoke an operation of another object, thus causing it to carry out an appropriate method, and perhaps return a value. Interestingly, one of the upshots of our hierarchical modeling of composite structure is that these interactions can be arranged to take on the form of either direct communication or broadcast.

The description in [HG] concentrates on a single-thread approach to 'event-driven' concurrency, which makes the exposition somewhat easier. For example, we need not concern ourselves there with the issue of multiple event queues. And while the presentation of both the syntax and semantics of the languages in [HG] is informal (for clarity) and not totally exhaustive (for brevity), we do have a full-fledged definition of both. This definition is reflected in the algorithm that

[3] A language is constructive if it contributes to the dynamic semantics of the model. That is, its constructs contain information needed in executing the model or in translating it into executable code. Other languages are reflective or assertive, and can be used by the system modeler to capture parts of the thinking that go into building the model — behavior included —, to derive and present views of the model, statically or during execution, or to set constraints on behavior in preparation for verification.

[4] The joint work we have been doing recently with the UML team, which is led by rational Corp. (see [Ra]), has resulted in full consistency between our object-model diagrams and the object models of UML. In particular, the semantics of composite objects is now defined in UML as it is in [HG].

RHAPSODY uses to translate any syntactically legal model into executable code. The current target language of RHAPSODY's code-synthesis is C++, but we hope that other languages will follow.

Spinoff research

Time permitting, this part of the talk will mention a few research topics we have been involved in, which were triggered in one way or another by the work on statecharts. Among them are (i) the question of succinctness of behavioral models (e.g., it is shown in [DH] that the kind of cooperative concurrency present in statecharts provides an exponential saving in size), (ii) the formalism of higraphs [H2], which is an extension and combination of graphs and hypergraphs, and (iii) the problem of drawing graphs and higraphs nicely [DaH, HS].

References

[B] Booch, G., *Object-Oriented Analysis and Design, with Applications* (2nd edn.), Benjamin/Cummings, 1994.

[CD] Cook, S. and J. Daniels, *Designing Object Systems: Object-Oriented Modelling with Syntropy*, Prentice Hall, New York, 1994.

[DaH] Davidson, R. and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", *ACM Transactions on Graphics* **15** (Oct. 1996), 301–331. (Also, Technical report, The Weizmann Institute of Science, Rehovot, Israel, 1989.)

[DH] Drusinsky, D. and D. Harel, "On the Power of Bounded Concurrency I: Finite Automata", *J. Assoc. Comput. Mach.* **41** (1994), 517–539.

[EGKP] Ehrig, H., R. Geisler, M. Klar and J. Padberg, "Horizontal and Vertical Structuring Techniques for Statecharts", *Proc. CONCUR '97*, to appear.

[H1] Harel, D., "Statecharts: A Visual Formalism for Complex Systems", *Sci. Comput. Prog.* **8** (1987), 231–274. (Preliminary version appeared as Tech. Report CS84-05, The Weizmann Institute of Science, Rehovot, Israel, Feb. 1984.)

[H2] Harel, D., "On Visual Formalisms", *Comm. ACM* **31** (1988), 514–530.

[H3] Harel, D., "Biting the Silver Bullet: Toward a Brighter Future for System Development", *Computer* (Jan. 1992), 8–20.

[HG] Harel, D. and E. Gery, "Executable Object Modeling with Statecharts", *Computer*, to appear. (Also, *Proc. 18th Int. Conf. Soft. Eng.*, Berlin, IEEE Press, March, 1996, pp. 246–257.)

[H+] Harel, D., H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", *IEEE Trans. Soft. Eng.* **16** (1990), 403–414. (Preliminary version appeared in *Proc. 10th Int. Conf. Soft. Eng.*, IEEE Press, New York, 1988, pp. 396–406.)

[HN] Harel, D. and A. Naamad, "The STATEMATE Semantics of Statecharts", *ACM Trans. Soft. Eng. Method.* **5**:4 (Oct. 1996), 293–333. (Preliminary version appeared as Tech. Report, i-Logix, Inc., 1989.)

[HPn] Harel, D. and A. Pnueli, "On the Development of Reactive Systems", in *Logics and Models of Concurrent Systems*, (K. R. Apt, ed.), NATO ASI Series, Vol. F-13, Springer-Verlag, New York, 1985, pp. 477-498.

[HP] Harel, D. and M. Politi, *Modeling Reactive Systems with Statecharts*, McGraw-Hill, to appear. (Early version titled *The Languages of* STATEMATE, Tech. Report, i-Logix, Inc. (250 pp.), 1991.)

[HPSS] Harel, D., A. Pnueli, J.P. Schmidt and R. Sherman, "On the Formal Semantics of Statecharts", *Proc. 2nd IEEE Symp. on Logic in Computer Science*, IEEE Press, New York, 1987, pp. 54–64.

[HS] Harel, D. and M. Sardas, "Randomized Graph Drawing with Heavy-Duty Preprocessing", *J. Visual Lang. and Comput.* **6** (1995), 233–253.

[HR] Huizing, C. and W. P. deRoever, "Introduction to design choices in the semantics of Statecharts", *Inf. Proc. Lett.* **37** (1991), 205–213.

[KP] Kesten, Y. and A. Pnueli, "Timed and Hybrid Statecharts and their Textual Representation", In *Formal Techniques in Real-Time and Fault-Tolerant Systems* (J. Vytopil, ed.), Lecture Notes in Computer Science, Vol. 571, Springer-Verlag, Berlin, 1992, pp. 591–619.

[L+] Leveson, N.G., M.P.E. Heimdahl, H. Hildreth and J.D. Reese, "Requirements Specification for Process-Control Systems", *IEEE Trans. Soft. Eng.* **20** (1995),684–707.

[MLPS] Mikk, E., Y. Lakhnech, C. Petersohn and M. Siegel, "On Formal Semantics of Statecharts as Supported by STATEMATE", Manuscript, 1996.

[M] Maraninchi, F., "Operational and Compositional Semantics of Asynchronous Automaton Compositions", *Proc. CONCUR '92*, Lecture Notes in Computer Science, Vol. 630, Springer-Verlag, Berlin, 1992, pp. 550–564.

[PU] Petersohn, C., and L. Urbina, "A Timed Semantics for the STATEMATE Implementation of Statecharts", Manuscript, 1996.

[PS] Pnueli, A. and M. Shalev, "What is in a Step: On the Semantics of Statecharts", *Proc. Symp. on Theoret. Aspects of Comput. Soft.*, Lecture Notes in Computer Science, Vol. 526, Springer-Verlag, Berlin, 1991, pp. 244–264.

[Ra] Rational Corp., Documents on UML (the Unified Modeling Language), Version 1.0 (http://www.rational.com/ot/uml/1.0/), 1996.

[R+] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[SGW] Selic, B., G. Gullekson and P. T. Ward, *Real-Time Object-Oriented Modeling*, John Wiley & Sons, New York, 1994.

[SM] Shlaer, S. and S. J. Mellor, *Object Lifecycles: Modeling the World in States*, Yourdon Press, 1992.

[vB] von der Beek, M., "A Comparison of Statechart Variants", in *Formal Techniques in Real-Time and Fault-Tolerant Systems* (Langmaack, de Roever and Vytopil, eds.), Lecture Notes in Computer Science, vol. 863, Springer-Verlag, New York, 1994, pp. 128–148.