Checking UML Dynamic Diagrams: A Synchronous Approach

C. ANDRÉ, M-A. PERALDI-FRATI, J-P. RIGAULT
Laboratoire Informatique Signaux et Systèmes (I3S)
Université de Nice Sophia Antipolis
CNRS UPRES-A 6070
Email {andre, map, jpr}@i3s.unice.fr

Abstract : This paper addresses the design of control-dominated systems using a synchronous approach and the UML. The work aims at formally checking the design: scenarios/controller consistency, safety properties. For this, a strengthening of UML behavioral models is necessary: SyncCharts are used instead of Statecharts, and Sequence Diagrams are extended with synchronously sound constructs akin to Message Sequence Charts. The formal foundations of the approach and the associated tools are briefly presented.

Keywords : Scenario, Statecharts, SyncCharts, Design, Model Checking.

Contact : C. André,
Laboratoire I3S, BP 121,
06903 Sophia Antipolis Cédex, France
Tél : +33. 4.92.94.27.11
Fax : +33. 4.92.94.28.96

Introduction

We are interested in the design of control-dominated systems as used in real-time. Our approach relies on synchronous programming [Be2000] and object-oriented modeling (UML).
In the UML, Sequence Diagrams and Statecharts [Ha87] are generally used for expressing dynamic behavior of objects and classes. Sequence Diagrams express scenarios, which are rather informal and constitute partial examples of system usage. Moreover they often "leave required properties about the intended system implicit" [vW98]. Statecharts are a state-based representation of class and object behaviors. Although they rely on formal semantics, the evaluation of their semantics is complex and may induce undesirable non-deterministic input/output behaviors. Note that the current UML definition does not provide any form of equivalence between these two types of diagrams.
Coming from the telecom community, Message Sequence Charts (MSCs) [RGG96] are a popular substitute for Sequence Diagrams; Their introduction in the UML has even been considered. They offer high level constructs like modularity, concurrency, iteration … There are many reports in the literature about analysis of MSC (*e.g.*, [AHP96]) and associated tools (*e.g.*, uBET from Lucent). However, there exist several semantic interpretations of MSC and each of the above mentioned works relies on a particular one. Choosing "a simple, yet expressive formal framework", Krüger et al [KGSB99] proposed an automated translation from MCS to Statecharts. This is a formal attempt to bridge the gap between the scenario-based and the state-based models.
The work presented in this paper adopts a similar type of translation between scenario and state-based model. It differs by choosing techniques and hypotheses that make it possible to check formal properties of the design.
Our underlying paradigm is the synchronous programming one [Be2000] which is based on a clear and deterministic semantics. The graphical companion model (SyncCharts[1] [An96]) is substituted for the UML Statecharts in order to express the state model. Concerning the expression of scenarios, we introduce synchronous-oriented enrichments to the Sequence Diagram model. These extensions, relying on a formal synchronous semantics, are suitable to express typical control situations and are liable to automated processing. Thus, model checkers like XEVE [Bo98] can be used to establish properties such as whether a scenario is executable or whether it is never possible.

The Synchronous Approach

Synchronous languages have been introduced to address the issue of reactive programming. The synchronous approach adopts an *abstract and ideal view* of a system. Interactions take place at discrete *instants*. *Simultaneity* of occurrences is an unambiguous concept. The synchronous paradigm relies on two main hypotheses:

[1] "SyncCharts" is the name of the model. A syncChart is an instance of this model.

Communications are supported by signals which are *instantaneously broadcasted*; Reactions are generated *without delay* and *in perfect synchrony* with the stimuli that triggered them. A noteworthy specificity of synchronous programming is the use of a multi-form logical time. Any event and especially repetitive ones may be considered as defining a time unit. Thanks to these strong hypotheses, synchronous languages have been given a clear and strict mathematical semantics so that the correctness of the design can be formally established.
Esterel is an imperative textual synchronous languages and SyncCharts is a graphical form of Esterel. "SyncCharts" is clearly inspired by Statecharts but differs in different aspects. The SyncCharts semantics is fully synchronous and perfectly fits Esterel's semantics. The semantics of Statecharts, such as the one adopted in Statemate, is more complex (micro-step semantics). Moreover, SyncCharts offers a richer expression of pre-emption. SyncCharts is now fully integrated in the Esterel commercial platform[2]. As a consequence, SyncCharts has direct access to the whole programming platform developed for Esterel: compilers, simulators (XES), model-checkers (XEVE), and circuit optimizers that rely on SIS and TiGeR (an efficient BDD-based tool).

Enrichment of UML Sequence Diagrams: the SIBs

Usually, Sequence Diagrams show the sequence of messages between objects. We have enriched the semantics of sequence at the object interface (corresponding to the events on the vertical line associated with an object). The model we propose, called *"Synchronous Interface Behavior"* (SIB), is a trade-off between complexity, expressiveness, and rigor. Basically, a SIB represents *a sequence of expected event occurrences* (signals in terms of synchronous modeling). Because SIB are partial observations of the system behavior, a set of *"observed events"* must be given. To capture time-related constraints or properties, some events are chosen as *"time-bases"* and their occurrences denote time passing. Binding logical time to real time can be done by relevant events (*e.g.*, a 1 kHz physical clock that generates signal "ms"). Multi-form (discrete) timing constraints are expressed by two special constructs: *window* and *before.*

- "p in window 1 .. 4 ms" says that expected actions "p" must occur between the first and the fourth future occurrence of ms. Note that replacing ms with meter is semantically perfectly correct and allows generalized forms of timeout: "stop_the_car in window 20 .. 30 meter".
- "do p before 8 .. 10 ms" means that expected actions "p" must occur before a delay of 8, 9 or 10 occurrences of ms has elapsed. Taking a lower bound different from the upper bound of the delay is a way to introduce a limited form of non determinism.

Synchronous modeling induces subtle issues, generally irrelevant to traditional approaches. Since simultaneity is meaningful, it is possible to expect a conjunction of event occurrences. Since instants are discrete, one can expect a stricly future occurrence of an event, or consider a possible present occurrence. Our notation captures these nuances.
A SIB generally represents more than a single expected sequence of events; it rather stands for a collection of sequences. Facilities are offered to express *variants* in sequences and *optional actions* (usually, optional expectations). The former leads to a partially ordered set of actions, instead of plain sequences. The latter requires the optional parts of scenarios to be delimited by observable event occurrences. A form of preemption (refer to the Esterel language) is used to specify such optional behaviors.
As most of synchronous formalisms, SIB has been given a *mathematical semantics*, in terms of conditional rewriting rules. Compared to Esterel or SyncCharts, SIB allows a certain degree of non determinism.
Thanks to this formal semantics, a SIB can be compiled into an equivalent synchronous module whose inputs are the "observed events" and that ends its execution by emitting either "Accepted", or "Not applicable".

Using SIBs

A SIB may be used for a better understanding of the behavior of the object, or for formally checking a property of the object. In both cases, the idea is to consider the SIB as a specification of an *observer*. In synchronous programming, an observer [Hal93] is a synchronous module run in parallel (synchronous composition) with the controller (the synchronous program to check). The SIB, or more accurately the associated module, observes the inputs and outputs of the controller. As soon as an unexpected event occurs, the SIB module terminates and emits "Not Applicable". If the reactions of the controller match the whole scenario, then the SIB module emits "Accepted".
The interpretation of "Accepted" and "Not applicable" depends on the type of property to check.
The simplest form is the "applicability of a scenario": Is the given SIB the input/output trace of a *possible* execution of the controller? To answer this question it is sufficient to show that "Accepted" is possibly emitted. Whenever "Not Applicable" is emitted, the considered input sequence must be given up, and another one is tried. This is just to see if the controller we have designed, can do what has been specified with the SIB.

[2] "Esterel Studio" marketed by Simulog.

More interesting is checking "safety properties". A safety property claims that a "bad news" will *never* occur. A way to establish a safety property is to verify that no sequence leading to a violation of the property is applicable. In this case, the "Accepted" signal is interpreted as the violation of the property. "Bounded responsiveness" (*i.e.*, an event B must occur in response to an event A, before the occurrence of an event C) is an instance of safety property often required in real-time applications. The *window* construct we have proposed is especially suited for expressing bounded responsiveness.
Simulation and Model Checking are two ways to validate a system. We use simulation for early bug finding. A first benefit of the translation of SIB into Esterel programs is the use of the interactive simulator (XES) provided with the Esterel distribution. As a designer, you can test scenarios through a user-friendly interface, and moreover, see the internal reactions of your controller. However, even if your design passes successfully all your simulation tests, you are not sure that a safety property holds. You need an exhaustive simulation of the controller behavior. Symbolic executions of the model can solve this problem. XEVE, the symbolic model checker, part of the Esterel distribution, does this job very well. A limitation is that signals must not convey values. This is the case for module associated with SIBs: they use only pure signals (associated with event occurrences) and counters. When a safety property is violated, XEVE generates a counter-example input sequence. This sequence can be played back with XES in order to understand the flaw.

Since SIBs have been given a semantics compatible with SyncCharts semantics, modules associated with SIBs can be composed as SyncCharts macro-states. A high-level SIB (HLSIB) is an arbitrary complex composition of SIBs using iteration, parallel composition and various preemptions. HLSIBs are at least as powerful as advanced MSCs and they are compilable into equivalent Esterel programs.

Conclusion

Our main objective is the design of safe controllers for critical reactive systems. Ambiguous models are disqualified for such applications. *Sequential Diagrams* (SDs) are often used to express expected behaviors of the controller. Since Sequential Diagrams are not given a clear semantics, there is no way to validtae the behavior they specify. To overcome this problem, we have proposed a substitute for SD, called *Synchronous Interface Behavior* (SIB). The semantics of SIB is mathematically defined in terms of a synchronous process algebra. SIBs can be composed with SyncCharts, a synchronous state-based graphical model. A behavior expressed with SIBs can be compiled into a semantically equivalent Esterel program. This program is then liable to validation either by interactive simulation (with XES) or by formal property checking (with XEVE).

The UML is now a standard methodology for the design of complex system. In order to use it in reactive and real-time system design, well-founded models are necessary for expressing the dymanic behavior of classes and objects. Real-time UML, such as proposed in Douglass's book [Do99], makes use of enriched Sequential Diagrams with stereotypes taylored to real-time applications. We propose to go further and adopt two models that rely on a strict semantics: SIB for scenarios and SyncCharts for based-state model.

Our approach has been tested on academic examples, so far. We still have to assess its scalability and its user-friendliness:

- *Scalability*: The current translation from SIBs into Esterel programs is a structural translation, with little optimization. The size of the generated code might be excessive for real-world systems. A second limitation is the size of the (symbolic) reachability set, which is used in property validations. The quantified timing constraints (*window* and *before* constructs) might cause rapid expansion of the state space.
- *User-Interface*: SIB and its semantics is strongly influenced by the synchronous approach. Some concepts may seem strange to users not familiar with the synchronous paradigm. A collection of typical examples should be available to convince potential users.

We believe that a better collaboration between the object paradigm (through UML) and the synchronous paradigm may facilitate the design of controllers and the formal verification of some of their properties.

References

[AHP96]R. Alur, G. J. Holzmann, and D. Peled. "An Analyzer for Message Sequence Charts."
Software Concepts and Tools, 17(2):70--77, 1996.
[AY99]R. Alur and M. Yannakakis. "Model checking of Message Sequence Charts." *10th Intern. Conference on Concurrency Theory,* Springer Verlag, 1999.
[An96] C. André. "Representation and Analysis of Reactive Behaviors: A Synchronous Approach" *IEEE-SMC Computational Engineering in Systems Applications (CESA),* Lille (F), July 1996, pp 19—29.

[Be2000] G. Berry “The Foundations of Esterel” in *Proof, Language, and Interaction,* Essays in honor of Robin Milner. G. Plotkin, C. Stearling, and M. Tofte, Editors. MIT Press, 2000.
[Bo98] A.Bouali. “Xeve: an Esterel Verification Environment.” *Int'l Conference on Computer-Aided Verification (CAV'98),* June/July 1998, Vancouver, BC Canada. Also available as a technical report INRIA RT-214, 1997.
[Do99] B. P. Douglass, “Doing Hard Time”, Object Technology Series, Addison-Wesley, 1999.
[Hal93] N. Halbwachs, “ Synchronous Programming of Reactive Systems”, Kluwer Academic Publishers, Amsterdam (NL), 1993.
[Ha87] D. Harel. “Statecharts: a Visual Formalism for Complex Systems”, *Science of Computer programming*, 1987, vol 8, pp 231-274.
[KGSB99] I. Krüger, R. Grosu, P. Scholz, and M. Broy. “From MCSs to Statecharts.” In Franz~J. Rammig, editor, *Distributed and Parallel Embedded Systems*, pages 61--71. Kluwer Academic Publishers, 1999.
[vW98] A. van Lamsweerde and L. Willemet. “Inferring declarative requirements specifications from operational scenarios”. *IEEE Tr. on Software Engineering*, 24(12):1089--1114, December 1998.
[RGG96]E. Rudolph, P. Graubmann, and J. Grabowski. “Tutorial on Message Sequence Charts” *Computer Networks and ISDN Systems*, 28(12):1629--1641, December 1996.