Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements*

David Harel, Hillel Kugler, and Amir Pnueli

Department of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Rehovot, Israel
{dharel,kugler,amir}@wisdom.weizmann.ac.il

Abstract. Constructing a program from a specification is a long-known general and fundamental problem. Besides its theoretical interest, this question also has practical implications, since finding good synthesis algorithms could bring about a major improvement in the reliable development of complex systems. In this paper we describe a methodology for synthesizing statechart models from scenario-based requirements. The requirements are given in the language of live sequence charts (LSCs), and may be played in directly from the GUI, and the resulting statecharts are of the object-oriented variant, as adopted in the UML. We have implemented our algorithms as part of the Play-Engine tool and the generated statechart model can then be executed using existing UML case tools.

1 Introduction

Constructing a program from a specification is a long-known general and fundamental problem. Besides its theoretical interest, this question also has practical implications, since finding good synthesis algorithms could bring about a major improvement in the reliable development of complex systems.

Scenario-based inter-object specifications (e.g., via live sequence charts) and state-based intra-object specifications (e.g., via statecharts) are two complementary ways to specify behavioral requirements. In our synthesis approach we aim to relate these different styles for specifying requirements. In [10] the first two coauthors of this paper suggested a synthesis approach using the scenario-based language of live sequence charts (LSCs) [7] as requirements, and synthesizing a state-based object system composed of a collection of finite state machines or statecharts. The main motivation for suggesting the use of LSCs as a requirement language in [10] is its enhanced expressive power. LSCs are an extension of message sequence charts (MSCs; or their UML variant, sequence diagrams) for rich inter-object specification. One of the main additions in LSCs is the notion of universal charts and hot, mandatory behavior, which, among other things, enables one to specify forbidden scenarios. Synthesis is considerably harder for

* This research was supported in part by the John von Neumann Minerva Center for the Verification of Reactive Systems, by the European Commission project OMEGA (IST-2001-33522) and by the Israel Science Foundation (grant No. 287/02-1).

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 309–324, 2005.

LSCs than for MSCs, and is tackled in [10] by defining consistency, showing that an entire LSC specification is consistent iff it is satisfiable by a state-based object system. A satisfying system is then synthesized.

There are several issues that have prevented the approach described in [10] from becoming a practical approach for developing complex reactive systems. A major obstacle is the high computational complexity of the synthesis algorithms, that does not allow scaling of the approach to large systems. Additional problems are more methodological, related to the level of detail required in the scenarios to allow meaningful synthesis, the problem of ensuring that the LSC requirements are exactly what the user intended, and a lack of tool support and integration with existing development approaches.

In this paper we revisit the idea of synthesizing statecharts from LSCs, with an aim of addressing the limitations of [10] mentioned above. Our approach benefits from the advances in research made since the publication of [10] – mainly the play-in/play-out approach [13], which supplies convenient ways to capture scenarios and execute them directly, and our previous work on smart play-out [11], which allows direct execution and analysis of LSCs using powerful verification techniques. We suggest a synthesis methodology that is not fully automatic but rather relies on user interaction and expertise to allow more efficient synthesis algorithms. One of the main principles we apply is that the specifier of the requirements provide enough detail and knowledge of the design to make the job easier for the synthesis algorithm. The algorithm tries to prove, using verification methods, that a certain synthesized model satisfies all requirements; if it manages to do so, it can safely synthesize the model. We have developed a prototype statechart synthesis environment, that receives as input LSCs from the Play-Engine tool [13] and generates a statechart model that can then be executed by Rhapsody [15], and in principle also by other UML tools, see e.g., [24, 27].

The paper is organized as follows. Section 2 describes the main challenges in synthesizing statecharts from scenarios and the main principles we adopt to address them. Section 3 shows how to relate the object model of LSCs as supported by the Play-Engine tool with standard UML object models, and describes how this is supported by our prototype tool. Section 4 addresses the notion of consistency of LSCs and introduces a game view for synthesizing reactive systems. Section 5 describes our approach to statechart synthesis, while Section 6 explains the actual statechart synthesis using an example of a cellular phone system. We conclude with a discussion of related work in Section 7.

2 Main Challenges in Synthesis

In this section we discuss some of the main challenges that need to be addressed in order to make a method for synthesizing statechart models from scenarios successful. The challenges are of different nature, varying from finding a scenario-based language that is powerful and easy for engineers to learn, to dealing with the inherent computational complexity of synthesis algorithms that must handle large complex systems.