Collaboration-Based Design of SDL Systems*

Frank Roessler[1], Birgit Geppert[1], and Reinhard Gotzhein[2]

[1] Avaya Inc., Software Technology Research
600-700 Mountain Ave., P.O. Box 636, Murray Hill, NJ 07974-0636, USA
{roessler,bgeppert}@avaya.com
[2] Computer Networks Group, Computer Science Department
University of Kaiserslautern
P.O. Box 3049, D-67653 Kaiserslautern, Germany
gotzhein@informatik.uni-kl.de

Abstract. The concept of *collaborations* capturing dynamic aspects of a distributed system across agent boundaries is elaborated in the context of SDL-2000. Several ways of composing collaborations are introduced, with collaborations being implicitly represented as SDL fragments. A new language for their *explicit* formal description, called CoSDL (**Co**llaborations in **SDL** systems) is then introduced and illustrated.

1 Introduction

The flexibility of protocol architectures is becoming more and more important as the pressure to deliver converged communication services to every corner of the globe grows. Due to ever growing demand of customers for new features, organisations are seeking for ways to reduce the time it takes to evolve their communications platforms and therefore reduce time to get new features to market. In this paper, we describe a new concept for structuring communicating state machines, which can help overcome these problems. Though the general approach works for the whole spectrum of automata based description techniques, we have yet focused on elaborating the details for the description and specification language SDL-2000 [17].

The research work underlying this paper originally aimed at formalising the SDL pattern approach [4,5,6] and providing adequate tool support. SDL patterns were introduced as reusable SDL artefacts accompanied with an incremental, scenario-driven design process. Though the relevance and usefulness of SDL patterns was demonstrated in many case studies, there was still an essential element missing: in order to be a true *construction set of protocol building blocks* (as envisioned in [5]), the approach needed a proper compositional framework for its reuse artefacts. When tackling the research problems regarding SDL pattern composition, soon it turned out that much broader architectural questions of an SDL system were involved.

* The work reported here was conducted while all authors were working at the University of Kaiserslautern, Germany.

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 72–89, 2001.

Many of the SDL patterns defined in [5] address interaction behaviour between two SDL agents. At first, we tried to treat such SDL patterns as components. This approach was not feasible, because SDL does not offer adequate language constructs for defining the subtle interface between an SDL pattern and its context specification. However, one can assemble individual SDL patterns to larger aggregates that have a much simpler interface. We call these aggregates *collaborations* and demonstrate how collaborations can be composed to complete SDL systems. Collaborations enable very flexible protocol architectures and are an important design principle for the development of SDL systems.

There are also two other viewpoints motivating the concept of collaborations independently from SDL patterns and showing that collaboration-based design is in fact a new approach to SDL system development that is independent of SDL patterns:

In general, interactions between communicating agents can always be described in two complementary ways: one of them centred on individual agents and the other focusing on a set of cooperating agents. Agents in SDL (as in other automata based description techniques such as Estelle [8], StateCharts [7], or ROOM [15]) are behavioural views that are deep but local. An SDL agent is precisely specified and immediately leads to executable code. However, it can be quite difficult to understand the overall functioning of an SDL system, because the behaviour of many agents must be analysed and combined for determining the entire system behaviour. To overcome this problem, a more holistic view on the behaviour of a collection of agents should be provided, and as shown in this paper, the concept of collaborations plays a key role here. Note that UML also supports this general notion of collaborations [2] and suggests class-, sequence-, and collaboration diagrams for their description. However, this paper develops the concept much further (in the context of SDL), as a much tighter integration of scenario and automata modelling is supported.

An important and complicated phase in distributed system development lies in the transition from system behaviour to the behaviour of interacting components. While the former is generally specified using scenario modelling techniques such as use case maps (UCM) [3], use cases [9], or message sequence charts (MSC) [16], the latter is typically captured by description techniques based on communicating extended finite state machines. It is commonly accepted that a systematic approach is required for this transition. One observation about these approaches is that scenario models are deliberately incomplete. They typically capture main system behaviour and important exceptional cases. While this adequately reflects and supports the process of understanding a distributed system from the perspective of a developer and maintainer, there is no reason other than complexity for not capturing all possible scenarios. In contrast to the scenario model, the automata model must eventually capture all possible scenarios. The problem is that these are not explicitly represented and that automata do not provide a natural way for designing scenario structures. We have explained this in the last paragraph and suggest that collaborations is a good solution for this.