SDL and Layered Systems: Proposed Extensions to SDL to Better Support the Design of Layered Systems

Rodolphe Arthaud

Telelogic France, 150 Rue Nicolas Vauquelin - BP 1310
31106 Toulouse Cedex – France
rodolphe.arthaud@telelogic.com

Abstract. Designing complex systems as stacks of collaborating layers is a common practice in various domains, from operating systems to user interfaces. It proves to be particularly fruitful in the domain of telecom systems and, more generally, in distributed systems. After showing why, today, SDL is not well suited for the design of layered systems, we explore usual techniques available in programming languages. Then, we attempt giving SDL the power of expression necessary to view and manipulate signals at different abstraction levels while preserving the language spirit, staying at design level.

1 Introduction

Complex systems have been thought of as independent communicating layers relatively early in various domains, such as operating systems [1], user interfaces (think of Xlib, Xt and Motif as layers) [4,9], protocols or a generic architecture such as the OSI Reference Model [3,8]. The concept of layers was one of the early design patterns [5], long before the concept of pattern was born as such. It was also one of the earliest techniques to favor reusability, maintainability and extensibility. A successful one too, since it could never be replaced by the newer and more fashionable concepts in the area — inheritance, type genericity, dynamic binding and other varieties. On the contrary, all these techniques only made it easier to think, design and program in terms of layers.

Since its origins, SDL has been used mainly in the context of telecom applications and, more specifically, for the design of protocols. Surprisingly enough, SDL does not offer any support for the design of *several, communicating* layers. As we will see, it is even weaker in that respect than mere low-level, programming languages.

In this paper, we will make a set of proposals for additions to SDL, to make it better suited for the design of multi-layered systems.

Please note that these proposals are not final ones, in the sense that similar power of expression could be obtained by different means, possibly with better results regarding static typing or consistency with other mechanisms of the

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 52–71, 2001.

language. In particular, a complete proposal might consider improving signal description together with data types. In this paper, we have chosen to focus on one topic only for the sake of clarity.

Please note also that the author does not claim that these ideas are original. On the contrary, they were inspired by existing languages and by discussions with SDL users. In particular, the author remembers some discussions within ITU-T meetings. This paper is an attempt to capture good ideas that could now be inserted in SDL.

Though these proposals should be extensions to SDL 2000, we have used the syntax of SDL 96 in the discussion for the sake of understandability, since we thought most readers would be more familiar with it. This has no impact on the discussion.

2 Presentation of the Problem

2.1 What Are Layers?

A layer is a set of components offering a certain service to higher-level layers and using lower-level layers.

Layers are highly independent from each other. In other words, the design imposed to the clients of a layer allows them to run on top of any implementation of this (lower) layer, and the layer itself can be used in the implementation of various applications. This is true of most component-based design.

We will say that a system consists of *layers* when it is split into communicating subsystems that work at *different levels of abstraction*. In particular, such subsystems usually share no data types apart from the most primitive ones; one subsystem is seen as more abstract than the other, closer to the application, to the problem space, and the other closer to the implementation level, to the solution space. In the rest of this paper, we will refer to *application* and *implementation* layers, though layered systems often have many more — but two will suffice for the discussion.

Two adjacent layers see a given piece of data in different ways; typically, what the application layer sees as a well-structured data type, is a mere string of bytes to the implementation layer.

In the context of communication, a layer is more or less transparent to its clients: that is, communicating processes may ignore that the messages they exchange go through the implementation layer — just as the author of a letter may ignore the details of mail transport as long as he or she knows how to write the address.

The lower-layer applies the same transformation to all data structures from the upper-layer. The receiver, after applying the reverse transformation to a piece of data, does not know what type it instantiates and how to interpret it. Some wrapping will be necessary, such as additional bytes in a header or additional leading and/or ending signals, carrying information such as type and length.

We hope that the following examples will clarify this.