Using SDL in a Stateless Environment*

Vassilios Courzakis[1], Martin von Löwis[2], and Ralf Schröder[2]

[1] Siemens AG, ICM N IS E12, Siemensdamm 50, 13629 Berlin
Vassilios.Courzakis@bln1.siemens.de
[2] Humboldt-Universität zu Berlin, Rudower Chaussee 25, 12489 Berlin
{loewis,schroed2}@informatik.hu-berlin.de

Abstract. Telecommunication services are often implemented to maintain their state in persistent storage. The application logic program then has no state variables of its own; it is seeded with a state depending on call context. Superficially, this contradicts the notion of extended state machines as they are defined by SDL-processes, where the state is part of the state machine. This paper presents an approach to separate state from program logic that is transparent to the SDL designer. This approach has been implemented in the SITE SDL runtime system in co-operation with Siemens, Berlin.

1 Introduction

Using SDL [1] to design and implement telecommunication services typically requires maintenance of per-call state information over the lifetime of the call. On a specific installation, many calls are processed both simultaneously (i.e. on multiple processors) and in an overlapping fashion. Therefore, a natural architecture is to create an SDL process every time a call is initiated or detected, and keep the per-call state in the variables of the process. Our experience shows that this implementation approach allows elegant definition of the program logic.

However, the approach has a number of drawbacks when reliability and load-sharing need to be considered. When translating SDL processes to target machine programs, process variables are typically translated to variables in a programming language for the target machine. Therefore, the per-call state will be maintained in the operating system process executing the protocol logic.

When assigning SDL processes to operating system processes, it is often not feasible to use a one-to-one mapping. Operating systems impose a limit on the number of processes they can create, and this limit is typically much smaller than the maximum number of concurrent calls that must be processed. As a result, many SDL processes are executed in the context of a single operating system process (potentially using operating system threads where possible). This scenario is shown in Fig. 1. When implementing an SDL system in such a manner, various aspects must be considered, and various strategies can be followed, see [2].

* We would like to thank our partners at Siemens, in particular Mr. Andreas Vogel, for the numerous ideas and suggestions that lead to the technology described in this paper.

R. Reed and J. Reed (Eds.): SDL 2001, LNCS 2078, pp. 316–327, 2001.

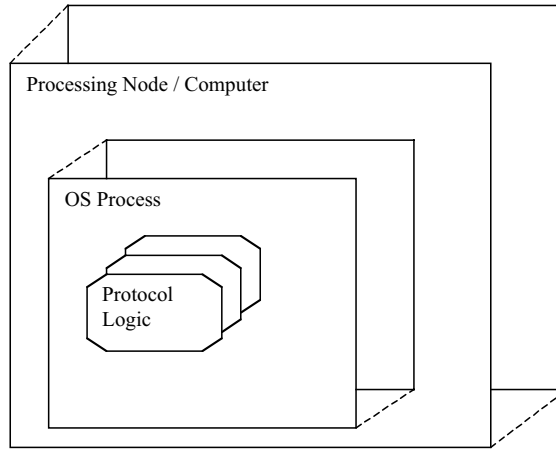

Fig. 1. Multiple SDL processes are executed in a single operating system process

To allow for load sharing, multiple processing nodes may be used to execute the same program logic. Likewise, multiple operating system processes may execute the program logic; such a scheme may be used to avoid the synchronisation that is common for multi-threaded runtime systems.

Unfortunately, the presented scheme for translating SDL processes into target system programs means that the association between SDL processes and operating system processes is fixed for the lifetime of the SDL process. The main reason is that the variables of each process are tied to the virtual address space of the executing process; moving them from one process to another is not supported by operating systems.

Likewise, reducing the number of available machines is difficult. Each operating system must be stopped before the machine can be switched off. In turn, each SDL process must stop before that, which means that each call must complete first. In applications with long-running calls, a significant amount of time may be needed to turn off the service on such a machine for maintenance.

Keeping the SDL process state in main memory also impacts the reliability and fault tolerance of the service. The potential failures range from hardware failures over operating system errors to errors in the SDL runtime system or the program logic. For many of these failures, not just the single SDL process will fail, but at least the entire operating system process, or the entire machine. That means that information on all calls that have been executed on that OS process or processing node is lost – even for calls that had no activity at the time of failure.

To provide fault tolerance, the state of a call must be stored persistently, or in a replicated way, or both. An SDL process whose state is stored persistently now needs to modify the persistent and replicated copy, instead of modifying the local OS virtual memory.

Computer systems typically provide a memory hierarchy, one layer of memory is a cache for the next layer of the hierarchy: for example, the main memory