Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 76 (2005) 15–28

The Journal of Systems and Software

www.elsevier.com/locate/jss

Smart debugging software architectural design in SDL

W. Eric Wong [a,*], Tatiana Sugeta [a,1], Yu Qi [a], Jose C. Maldonado [b]

[a] *Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, United States*
[b] *Department of Computer Science, University of Sao Paulo at Sao Carlos, Sao Carlos, SP, Brazil*

Received 15 December 2003; received in revised form 15 April 2004; accepted 15 June 2004
Available online 11 September 2004

Abstract

Statistical data show that it is much cheaper to fix software bugs at the early design stage than the late stage of the development process where the final system has already been implemented and integrated together. The use of slicing and execution histories as an aid in software debugging is well established for programming languages like C and C++; however, it is rarely applied in the field of software specification for designs. We propose a solution by applying the technology at source code level to debugging software designs represented in a high-level specification and description language such as SDL. More specifically, we extend execution slice-based heuristics from source code-based debugging to the software design specification level. Suspicious locations in an SDL specification are prioritized based on their likelihood of containing faults. Locations with a higher priority should be examined first rather than those with a lower priority as the former are more suspicious than the latter, i.e., more likely to contain the faults. A debugging tool, SmartD$_{SDL}$, with user-friendly interfaces was developed to support our method. An experiment is reported to demonstrate the feasibility of using our method to effectively debug an architectural design.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Architectural design; SDL; Testing; Debugging; Execution slice; Fault detection; SmartD$_{SDL}$

1. Introduction

Statistical data show that it is much cheaper to fix software bugs at the early design stage than the late stage of the development process where the final system has already been implemented and integrated together. An ideal scenario would be to conduct as much testing on design as possible followed by an effective debugging to locate all the bugs, if any, so that they can be removed before the coding is started. With this in mind, two important questions have to be answered: how to represent an architectural design so that it can be easily tested and debugged, and what kind of debugging methods should be used to help programmer effectively locate the bugs in an architectural design.

In this paper, the architectural design of a software system is represented by SDL (a specification and description language) for the following reasons. SDL satisfies the requirements for an executable architecture description language (Luckham et al., 1995). It allows dynamic creation and termination of process instances and their corresponding communication paths during execution. As a result, SDL is capable of modeling the architectures of dynamic distributed systems in which the number of components and connectors may vary during system execution. In addition, SDL can represent all four views of software architectures (Kruchten, 1995). For example, SDL uses delay and non-delay channels to indicate the relative physical locations of components (Belina et al., 1991). Moreover, since SDL specifications are executable, its dynamic execution trace

[*] Corresponding author. Tel.: +1 972 883 6619; fax: +1 972 883 2349/2399.
E-mail address: ewong@utdallas.edu (W.E. Wong).
URL: http://www.utdallas.edu/~ewong.
[1] Miss Tatiana Sugeta is a visiting student from the University of Sao Paulo at Sao Carlos.

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.06.026

(including the exact execution counts of a given part of an SDL specification, such as a decision, a state input, or a context variable) generated during the simulation [2] can be collected. An architectural design in SDL-92 can be viewed, for example, as a collection of blocks [3] and processes communicating with each other by exchanging signals through channels (Ellsberger et al., 1997). The top level is a system level specification. Each system contains some blocks; each block contains either blocks or processes; blocks communicate through channels; each channel can be either delaying or non-delaying; each process of a block is defined by an extended finite state machine (EFSM); they communicate through channels. Altogether, an SDL specification provides a process view of a system's architectural design.

The textual representation of an SDL specification can be viewed as "a program" in SDL, just like a program in C. As a result, all the debugging methods applied to C programs can also be applied to SDL specifications. In particular, we are interested in using execution slice-based heuristics for locating faults [4] in an SDL specification. Under this scenario, an execution slice with respect to a given test case contains the set of SDL code executed by this test. More specifically, we can also represent an execution slice as a set of blocks, decisions, c-uses, or p-uses, respectively, with respect to the corresponding block, decision, c-use, or p-use coverage criterion. Below we provide a brief explanation on a block defined in block coverage and a decision defined in decision coverage to clarify some possible confusion.

The block coverage for SDL specifications is similar to the well-known basic block coverage for C programs where a basic block, also known as a block, is a sequence of consecutive statements or expressions containing no transfers of control except at the end, so that if one element of it is executed, all are. This of course assumes that the underlying hardware does not fail during the execution of a block. As for SDL, the "block" in the "block coverage" means a subset of an SDL specification that will always be executed together (Wong et al., 2003). It has a different meaning from the "block structure" in SDL which represents either a local specification within a system specification or a remote specification to which the system specification must contain a reference, which then becomes a specification of the next abstraction level. Although this might cause some inconvenience, there should be no ambiguity about which "*block*" is referred to, i.e., whether it means a block in block coverage or a block as a structuring element in SDL, after the surrounding context is considered. Hereafter, we refer to "block" as the former unless otherwise specified. The "decision" coverage in SDL considers not only "data-related decision" but also "event alternatives" (Wong et al., 2003). That is, this criterion considers more than just the decision construct represented by a diamond. In fact, the branching of blocks can be caused either by the state transitions due to different inputs or by decision matching. However, to be consistent with the name (i.e., decision coverage) used for C programs, we name this criterion also as "decision" coverage criterion. As for the c-use and p-use criteria for SDL, they are very similar to those for C programs. Interested readers can refer to Ural et al. (2000) and Wong et al. (2003) for more details.

The objective of our study is to apply the technology at source code level to debugging software designs represented in a high-level specification and description language such as SDL. Execution slice-based heuristics are extended from source code-based debugging to the software design specification level. Suspicious locations in an SDL specification are prioritized based on their likelihood of containing faults. Locations with a higher priority are more suspicious and more likely to contain faults than those with a lower priority. A debugging tool, $\text{SmartD}_{\text{SDL}}$, with user-friendly interfaces was developed to support our method. An illustration on how this tool can help programmers in debugging is also provided.

The rest of the paper is organized as follows. Section 2 describes an execution slice-based approach for debugging. A debugging tool for SDL, $\text{SmartD}_{\text{SDL}}$, and the underlying methodology used for implementing this tool, is explained in Section 3. An experiment of applying our methodology to the SDL specification of an IGCS call agent appears in Section 4. Section 5 discusses how to develop an effective debugging strategy. Section 6 presents an overview of some related studies. Finally, in Section 7 we offer our conclusions and recommendations for future research.

2. An execution slice-based approach for debugging

We first present a brief overview of the commonly used program slicing: static, dynamic, and execution. For more details, interested readers should refer to the references listed in Section 2.1. Then, we describe

[2] In this paper, "executing an SDL specification" has the same meaning as "simulating an SDL specification." We also use "SDL specifications" and "SDL code" interchangeably. In addition, since we use Telelogic Tau as the simulator for generating execution trace with respect to each test case, the version of the SDL used by our debugging tool, $\text{SmartD}_{\text{SDL}}$ (refer to Section 3) has to be consistent with that supported by Tau. With this in mind, we used SDL-92 instead of SDL-2000.

[3] Here, a block is a structuring element in SDL. It is different from a block defined for the block coverage criterion. Refer to the subsequent paragraphs for more details.

[4] "Faults" and "bugs" are used interchangeably.

Download English Version:

https://daneshyari.com/en/article/10348911

Download Persian Version:

https://daneshyari.com/article/10348911

Daneshyari.com