Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 111 (2005) 27–52
www.elsevier.com/locate/entcs

Architectural Unit Testing

Giuseppe Scollo and Silvia Zecchini[1,2]

Department of Computer Science
University of Verona
Verona, Italy

Abstract

A formal testing methodology is outlined in this paper, that proves applicable to validation of architectural units in object-oriented models, and its use is illustrated in the context of the design of a robot teleoperation architecture. Automated generation of test cases to validate the functionality of the robot trajectory generation unit showcases the key features of this methodology. A disciplined use of UML state diagrams, to model the unit's dynamics consistently with its static properties as modeled by class diagrams, enables one to provide such models with Input/Output Labelled Transition Systems (IOLTS) semantics, whence a rich machinery of testing theories and tools based on those theories become readily available. Our case study tells that, besides black-box testing of final implementation units, white-box analysis of architectural units may greatly benefit from the flexibility of parameterized I/O-conformance relations. Test purposes turn out to be a useful methodological link between functional requirements, which they are drawn from, and conformance relations, which they help one to instantiate, thereby delimiting test selection to purposeful tests. Contingent aspects of our methodology include: a mechanical translation of state diagrams in Basic LOTOS, a non-mechanical, use-case driven synthesis of test purposes, expressed in the same language, and the use of the TGV tool for automated test case generation. Other choices in these respects are well possible, without affecting the characteristic traits of the proposed methodology, that are rather to be found in: 1) the combination of object-oriented architectural modeling with IOLTS semantics; 2) the aim at maximizing the potential for test generation from UML models, in a broad view of testing which applies throughout the development process; 3) the specific proposal to consider internal actions as testable actions, in view of a better coordination between testing (discovery of faults) and debugging (discovery of internal sources of faults).

Keywords: formal testing methods, white-box testing, test purpose, test selection, automated test case generation.

[1] Email: giuseppe.scollo@univr.it
[2] Email: silvia.zecchini@univr.it

doi:10.1016/j.entcs.2004.12.006

1 Introduction

The analysis, design and construction of a complex system can be made conceptually more tractable if one describes the software architecture by a formal specification [21]. A specification of such type enables engineers and designers to check which components' functionalities, described in the system requirements, are satisfied and to verify the intended interactions of those components. The formal specification of a software architecture provides a solid foundation for developing architecture-based testing techniques [22].

The testing of architectural abstractions allows one to detect defects in the initial phases of the software lifecycle, rather than after implementation or during system integration, as is common practice, and thus to prevent their propagation through the subsequent phases.

In very complex software systems, the amount of information in the system implementation is, typically, more than a single person could understand. A common way to deal with these systems is by using a *model* of the system. The availability of a model derives, obviously, from the application to realize. Clearly, in a model we must include all the relevant information for our purpose, but we must pay attention to exclude the information that is not necessary. Indeed, a model with too much information may be difficult to comprehend. The name "*model-based testing*" is a general term used to refer to an approach that bases testing activities, such as test case generation and evaluation, on models of the application under test [6,1,4].

Object-oriented models have found in the Unified Modeling Language (UML) [20] a standard notation, supported by a wide variety of model development tools. This enables one to model design concerns, requirements as well as decisions, at different abstraction levels, or *perspectives* [3], ranging from the *conceptual* modeling perspective through a more prescriptive *specification* perspective, down to concrete *implementation* perspective. Clearly, all of these prove useful, albeit in different phases of the software development process, but we argue that there's even room in between. Of particular interest to this paper is an *architectural* perspective, which is more prescriptive than conceptual modeling in that it fixes design decisions of architectural relevance such as naming of components (packages, classes) and connectors (associations, operations, inheritance relations), as well as ordering of interactions between objects, yet not so complete in its prescriptive character as a specification perspective would be.

In the next section we characterize with some more precision the level of formal detail which is adopted in the architectural perspective taken in the subject case study. For the time being we just point out that several types of