A Specification-Driven Interpreter for Testing Asynchronous Creol Components[⋆]

3. October 2008

Marcel Kyas[1] and Andries Stam[2] and Martin Steffen[1] and Arild B. Torjusen[1]

[1] University of Oslo, Norway
[2] Almende, The Netherlands

1 Motivation

Software testing [12] is an established practice to ensure the quality of programs and systems. Hosts of different testing approaches and frameworks have be proposed and put to (good) use over the years. Formal methods and program language theory have proven valuable to render testing practice a more formal, systematic discipline (cf. e.g. [5]). In itself not a new proposal —first inspirations to put testing on more formal grounds can be dated back as early as the seminal Nato conference on "Software Engineering" [8]— formal approaches to testing have has gained momentum in recent years, as for instance witnessed by the trend towards model-based testing [4,1]. In this paper we propose and explore a formal approach for black-box testing asynchronously communicating components in open environments.

Creol: a language for asynchronously communicating, active objects Creol [3,7] is a high-level, object-oriented language for distributed systems, featuring active objects. Creol is formally defined and especially its operational semantics is implemented in rewriting logic, using Maude [2] as execution platform. Its communication model is based on exchanging messages *asynchronously.* This is in contrast with object-oriented languages based on multi-threading, such as *Java* or $C^\sharp$, which use "synchronous" message passing in which the calling thread inside one object blocks and control is transferred to the callee. Exchanging messages asynchronously decouples caller and callee, which makes that mode of communication advantageous in a distributed setting. On the other hand, the asynchronicity makes validating and testing of programs more challenging.

Behavioral interface description language Abstracting from internal executions, the black-box behavior of components is given by interactions at their

⋆ Part of this work has been supported by the EU-project IST-33826 *Credo: Modeling and analysis of evolutionary structures for distributed services* and the German-Norwegian DAAD-NWO exchange project *Avabi* (Automated validation for behavioral interfaces of asynchronous active objects).

interface. We formalize the interface specification language over communication trace labels to specify components in terms of traces of observable behavior.

In the specification language, a clean separation of concerns between interaction under the control of the component or coming from the environment is central, which leads to an assumption-commitment style description of a component's behavior. The assumptions *schedule* the order of inputs, whereas the outputs as commitments are being *tested* for conformance. To ensure the mentioned separation of responsibilities, we define well-formedness conditions which in addition assure that only "meaningful" traces, i.e., those corresponding to actual behavior, can be specified. The specification language is characterized by two other salient features: it allows to specify freshness of communicated values and furthermore, it respects the asynchronous nature of communication in Creol: Due to asynchronous communication, the order in which outgoing messages from a component are observed by an external observer does not necessarily reflect the order in which they where actually sent. We take this into account by only considering trace specifications up-to, an appropriate notion of *observational equivalence.* The specification language is a simple recursive trace language designating sets of finite traces over communication labels.

A second point to stress is that the specification language is designed to be efficiently executable on Creol's executing platform and thus be used for testing a component. We define an operational semantics for the specification language. Th is done by synchronising the execution of the specifications with that of the component for the purpose of both generating the required input to the component and at the same time testing that the output behavior of the component conforms to the specification, up-to observational equivalence.

2 Results

The paper extends the technical report [6], which concentrates on the formalization, and contains the following contributions:

Formalization: We formalize the interface behavior of a concurrent, object-oriented, language plus a corresponding behavioral interface specification language. This gives the basis testing active Creol objects, where a test environment can be simulated by execution of the specifications.

Implementation: The existing Creol interpreter, realized by rewriting logic on the Maude platform, is extended with the implementation of the mentioned specification language. This yields a specification-driven interpreter for testing asynchronous Creol components.

Case study: As a case study, we apply the test methodology to a model of an industrial software system which is inherently multi-threaded and based on asynchronous communication.

References

1. *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.
2. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In Nieuwenhuis [11], pages 76–87.
3. The Creol language. `http:heim.ifi.uio.no/creol`, 2007.
4. S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 1999 International Conference on Software Engineering, 1999*, pages 285–294, 1999.
5. M.-C. Gaudel. Testing can be formal, too. In Mosses et al. [9], pages 82–96.
6. I. Grabe, M. Steffen, and A. B. Torjusen. Executable interface specifications for testing asynchronous Creol components. Technical Report 375, University of Oslo, Dept. of Computer Science, July 2008. A shorter version has been submitted for conference proceedings.
7. E. B. Johnsen, O. Owe, and I. C. Yu. Creol: A type-safe object-oriented model for distributed concurrent systems. *Theoretical Computer Science*, 365(1–2):23–66, Nov. 2006.
8. A. I. Llewelyn and R. F. Wickens. The testing of computer software. In Naur and Randell [10], pages 189–199.
9. P. D. Mosses, M. Nielsen, and M. I. Schwarzbach, editors. *TAPSOFT '95: Theory and Practice of Software Development, 6th International Joint Conference CAAP/FASE*, volume 915 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.
10. P. Naur and B. Randell, editors. *Software Engineering: A Report on a Conference sponsored by the NATO science committee*. NATO, Jan. 1969.
11. R. Nieuwenhuis, editor. *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA 2003)*, volume 2706 of *Lecture Notes in Computer Science*. Springer-Verlag, June 2003.
12. R. Patton. *Software Testing*. SAMS, second edition, July 2005.