Communicating Process Architectures 2003
Jan F. Broenink and Gerald H. Hilderink (Eds.)
IOS Press, 2003

Tutorial: Prioritised Service Architecture using Honeysuckle

Ian R. EAST
Department of Computing, School of Technology
Oxford Brookes University, Wheatley Campus, Oxford, England OX33 1HX
ireast@brookes.ac.uk

Abstract. An update will be presented on the progress in establishing the Honeysuckle programming language [1] and its formal foundations. The latter are formally addressed in a paper currently under review for journal publication [2], but will be summarised. They include formal definitions of service protocol plus service network/component (SNC) and the PSA design rule (PSADR), from which a proof of a priori deadlock-freedom emerges directly. Freedom from priority conflict (and thus inversion) is also easily guaranteed. Closure in the definition of system and component guarantees true compositionality under both concurrency and prioritised alternation.

Summary

1. Introduction

The dramatic fall in hardware cost is leading to new software application domains that require the combination of high integrity and low development cost. Traditionally, high software integrity is attained through the use of a posteriori verification of formal specifications using additional development tools. Combined mathematical and programming skills are thus required, which are in short supply. Furthermore, such formal methods do not scale well to large applications. This should not be surprising since they substitute "trial-and-error" for true engineering. Finally, the implied additional cost and risk cannot easily be justified for smaller applications.

An alternative is to employ formal design rules and a priori (static) verification. Such rules can be built into the definition of a programming language and verification built into an appropriate compiler. As a result, no additional tools or skills are required and the method may be expected to scale well.

A second problem addressed is the absence of compositionality at the design level in both process- and object-oriented languages. The specification of any system must remain uncompromised by the substitution of one component by another with the same interface. Achieving compositionality under concurrency while denying the possibility of interference remains a goal in computer science [3]. Here, the primary aim is at least to deny the worst pathological behaviour, namely deadlock, using known effective design rules [4, 5].

Honeysuckle attempts to simultaneously eliminate the possibility of deadlock while programming concurrency and yield true compositionality without complicating abstraction. Human abstraction of the every-day world includes concurrency and prioritized alter-

nation from an early age yet their application to software remains a task for the élite. Honeysuckle is intended to demonstrate that, with the right model for abstraction, it need not be so.

1.1 Service architecture and deadlock-freedom

Proof of *a priori* deadlock-freedom already exists [5], subject to a simple design rule whose verification may be performed at compile-time. However, *ad hoc* conditions are employed. It is shown how appropriate definitions of *service* and *service network* can incorporate all necessary conditions and thus permit a proof of deadlock-freedom free of any additional constraint.

Extending the definition of service network to allow mutual exclusion and dependency of service is shown to achieve closure between service network and service network component (SNC). An adapted proof of deadlock-freedom is then possible, subject to a revised design rule. Recursive parallel (de)composition of systems then becomes possible, retaining deadlock-freedom without need of further proof.

1.2 Prioritised service architecture, compositionality, and abstraction

The same extension is shown also to afford secure (de)composition under prioritized alternation, and a further design rule to ensure freedom from priority conflict. The usual restriction of service (client/server) networks to tree structure need thus not apply.

PSA may prove sufficient alone to express the specification of many systems, negating the need for any meta-level. It is particularly well suited to systems that are distributed and/or embedded. A prioritised service interface (PSI) captures input/output dependency, mutual exclusion between inputs, and their prioritization. It is not limited to listing input and output connections. It can also express how they *relate*.

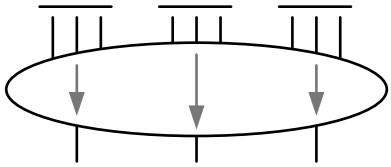

Figure 1 A (black-box) system specified by its prioritised service interface (PSI).

True design-level compositionality requires closure in the definition of system and component, which yields the model for abstracting an interface to either. A PSA design is expressed via an appropriate prioritized service digraph (PSD), which comprises two distinct species of arc which may be drawn in different colours (*e.g.* blue for service, red for interruption).

Process-oriented programming is entirely compatible with object-oriented programming (OOP), though the reverse is an assertion that is less easily supported. There are a number of distinct models for OOP of concurrency, which is usually considered an advanced topic [6, Chapter 30]. (Manfred Broy has recently contributed a general critique of OOP [7].)

In Honeysuckle, the relationship between process and object is simple. Processes *own* objects. Ownership may be transferred between concurrent processes. This mirrors the temporary transfer that takes place between processes in sequence when a parameter is passed by reference to a procedure. (In occam, only values, and not objects, may be passed across channels – a limitation which can give rise to inefficiency through otherwise unnecessary copying.) References to value and object are distinguished implicitly, by operator, avoiding the long-understood problems associated with explicit references (pointers) [8].

1.3 Conclusion

Honeysuckle is intended for the specification, compositional design, and implementation, of concurrent and reactive systems with inherent security against deadlock. In the use of design rules, prioritized service architecture provides for the engineering of systems without additional tools or scarce combination of skills. It should thus be suitable for the affordable development of new computer applications that demand high integrity at low cost (and risk).

A new and consistent formal model for program abstraction exists in the form of prioritized service architecture (PSA). Development tools can easily verify adherence to design rules as the system is incrementally composed *or modified*. (It should be remembered that ~98% of programmer activity is in modifying existing systems.) Thus formal guarantees can be made regarding behaviour *a priori*.

PSA is adequate alone for the specification of many systems and affords true compositionality. Abstraction is natural and powerful, facilitating straightforward capture of the concurrent and reactive behaviour typical of embedded, distributed, and interactive systems.

The complete specification for Honeysuckle will be published openly on the internet. Funding is being sought for the development of a compiler and interactive development environment. Development tools will also be “open source”. It is then intended to demonstrate the advantages of the method, perhaps by shadowing a commercial project.

Further work is also planned to investigate the semantics of the language and whether additional guarantees can be made regarding behaviour, for example, freedom from livelock. Another possibility is to allow the introduction *ad hoc* of an additional rule known in advance to guarantee some application-specific safety constraint. The circumstances under which interference can be eliminated, without compromising efficiency of implementation, is also of obvious interest.

2. Acknowledgements

The author acknowledges many valuable, and thoroughly enjoyable discussions with Jeremy Martin, whose D.Phil. thesis addressed many of these issues, and Sabah Jassim, who shared with me the enviable task of supervision.

References

1. East, I.R., The Honeysuckle Programming Language: An Overview. IEE Software, 2003. 150(2), pp. 95-107.
2. East, I. R., Deadlock-Free Programming with Service Protocol, submitted to ACM ToPLAS, 2003.
3. Jones, C.B., Wanted: A Compositional model for Concurrency, in Programming Methodology, McIver and C. Morgan, Editors. 2002, Springer-Verlag. pp. 1-15.
4. Martin, J., I. East, and S. Jassim, Design Rules for Deadlock Freedom. Transputer Communications, 1994. 2(3): pp. 121-133.
5. Martin, J.M.R., The Design and Construction of Deadlock-Free Concurrent Systems. 1996, University of Buckingham: Hunter Street, Buckingham MK18 1EG.
6. Meyer, B., Object-Oriented Software Construction. 1997: Prentice-Hall.
7. Broy, M., Object-Oriented Programming and Software Development – A Critical Assessment, in Programming Methodology, A. McIver and C. Morgan, Editors. 2003, Springer. pp. 211-221.
8. Hoare, C. A. R., Recursive Data Structures. Int. J. Computer and Information Sciences, 1975, 4(2), pp. 105-32. Reprinted as Chapter 14 in [9].
9. Hoare, C. A. R., Essays in Computing Science. 1989: Prentice Hall