Models and Calculi for Software Components

Tartu University
December 2003

Luís Soares Barbosa
(lsbdi.uminho.pt)

Departamento de Informática
Escola de Engenharia
Universidade do Minho
Braga, Portugal

1 Context

An increasing number of computer based systems are based on the cooperation of distributed, heterogeneous components organised into open software architectures that, moreover, can survive in loosely-coupled environments and be easily adapted to changing application requirements. Such is the case, for example, of applications designed to take advantage of the increased computational power provided by massively parallel systems or of the whole business of Internet-based software development. In order to develop such systems in a systematic way, the focus in development methods has switched, along the last decade, from functional to structural issues: both data and processes are encapsulated into software units which are connected into large systems resorting to a number of techniques intended to support reusability and modifiability. This encapsulation principle is essential to both the *object-oriented* and the more recent *component-based* software engineering paradigms.

In fact, although the term *software component* has been around for a long time, *component-based programming* has become a buzzword since mid 1990's (see, *e.g.*, [10, 12, 14, 13, 9, 15]). The basic motivation is to replace conventional programming by system's construction by composition and configuration of reusable off-the-shelf units, often regarded as *'abstractions with plugs'*. In fact all engineering disciplines rely on standard components to design and build their artifacts — software engineering should not be the exception.

A *component* is a 'black-box' entity which both provides and requires services, encapsulated through a public interface which may exhibit both static and behavioural information. Reasoning about components brings to scene two other elements: the *software architecture* [7, 1, 6] defining the connectors and corresponding composition rules and the *script* which specifies how components are plugged together, therefore making architectures explicit. If components are regarded as active units in a distributed (or, at least, concurrent) environment, scripts are called *coordination models*. The underlying paradigm also leads to think in terms of *families* of systems (rather than single systems) and component *taxonomies*, making it easier to catalogue components and retrieve them from suitable repositories.

As it happened before with *object* orientation, *component programming* has grown up to popular technologies before consensual definitions and principles, let alone formal foundations, have been put forward. Actually, recent years have seen the emergence of techniques and technologies for the specification and implementation of component-based software architectures. Middleware platforms have proposed a variety of component models (from Omg's Corba to implementations of IBM's and SUN's EJB or MICROSOFT's .NET). Such technologies are designed around specific *interaction paradigms* — *e.g.*, CORBA and JAVABEANS based on object-like composition, JAVASPACES and TSPACES on top of *shared data space* models, typical of classical coordination languages, and *channel* based techniques inherited from process languages, as in IWIM/MANIFOLD or DRAWIN. They also differ at the application level — *e.g.* acting as source-level languages extensions (such as CORBA or JAVASPACES), or as connector frameworks for dynamic (binary-level) units (as in *e.g.*, COM and .NET). Similarly, metamodels have been developed and standardised to specify component based software architectures (e.g., UML 2 or the EDOC UML specialisation).

These approaches provide the ability to clearly model and organise components into architectures. However, they allow limited space for formal analysis and reasoning (both at the local and global levels).

2 Objectives and Plan

Although the term 'software component´ has been around for a long time, component-based programming has become a buzzword since mid 1990's. The basic motivation is to replace conventional programming by system's construction through composition and configuration of reusable off-the-shelf units, often regarded as 'abstractions with plugs'. Actually all engineering disciplines rely on standard components to design and build their artifacts — software engineering should not be the exception. However, as it happened before with object orientation, component programming has grown up to popular technologies before consensual definitions and principles, let alone formal foundations, have been put forward.

These lectures will address a number of issues in the development of semantical models and *calculi* for software components and component composition. Two such models will be presented: a calculus of generalized Mealy machines parametric on a behaviour specification, introduced as a strong monad, and a relational model for software connectors.

Both models are framed on a coalgebraic approach to semantics. What does refinement mean in such a setting will be a major topic for discussion.

The course includes an overview of coalgebra theory and coinduction, through a case study on the 'reconstruction´ of classical process algebra in an essentially pointfree, calculational style.

The course material, based on recent results and work in progress (see, *e.g.*, [2, 3, 4, 5]), is framed on a *coalgebraic* approach to semantics [8, 11]. Some constructions will be illustrated by resorting to a functional language as a prototyping vehicle.

Plan

- **8th Dec** Introduction to coalgebraic reasoning (`sLec1.ps`)
- **8th Dec** Processes as coalgebras (`sLec2.ps, sLec3.ps`)
- **10th Dec** Components as coalgebras (`sLec4.ps`)
- **12th Dec** Refinement (`sLec5.ps`)
- **12th Dec** Connectors as relations (`sLec6.ps`)

The course material also includes some notes on *monads* (`monads.ps`), *bicategories* (`bicat.ps`) and a few exercises (`exerc.ps`).

References

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM TOSEM*, 6(3):213–249, 1997.

[2] L. S. Barbosa. Components as processes: An exercise in coalgebraic modeling. In S. F. Smith and C. L. Talcott, editors, *FMOODS'2000 - Formal Methods for Open Object-Oriented Distributed Systems*, pages 397–417. Kluwer Academic Publishers, September 2000.

[3] L. S. Barbosa. Process calculi *à la* Bird-Meertens. In *CMCS'01 - Workshop on Coalgebraic Methods in Computer Science*, pages 47–66, Genova, April 2001. ENTCS, volume 44.4, Elsevier.

[4] L. S. Barbosa and J. N. Oliveira. Coinductive interpreters for process calculi. In *Proc. of FLOPS'02*, pages 183–197, Aizu, Japan, September 2002. Springer Lect. Notes Comp. Sci. (2441).

[5] L. S. Barbosa and J. N. Oliveira. State-based components made generic. In H. Peter Gumm, editor, *Elect. Notes in Theor. Comp. Sci. (CMCS'03 - Workshop on Coalgebraic Methods in Computer Science)*, volume 82.1, Warsaw, April 2003.

[6] J. Fiadeiro and A. Lopes. Semantics of architectural connectors. In *Proc. of TAPSOFT'97*, pages 505–519. Springer Lect. Notes Comp. Sci. (1214), 1997.

[7] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering (volume I)*. World Scientific Publishing Co., 1993.

[8] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[9] B. Meyer and C. Mingins. Component-based development: From buzz to spark. *IEEE Computer*, 32(7):35–37, 1999.

[10] O. Nierstrasz and L. Dami. Component-oriented software technology. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, pages 3–28. Prentice-Hall International, 1995.

[11] J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249(1):3–80, 2000. (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[12] J. Sametinger. *Software Engineering with Reusable Components*. Springer-Verlag, 1997.

[13] J.-G. Schneider and O. Nierstrasz. Components, scripts, glue. In L. Barroca, J. Hall, and P. Hall, editors, *Software Architectures - Advances and Applications*, pages 13–25. Springer-Verlag, 1999.

[14] C. Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[15] P. Wadler and K. Weihe. Component-based programming under different paradigms. Technical report, Report on the Dagstuhl Seminar 99081, February 1999.