MFES: Software Architecture for Reactive Systems

Luís Soares Barbosa

A roadmap for the 2011-12 course edition

1 General course description

Software architecture. Software architecture emerged as a proper discipline in Software Engineering, from the need to explicitly consider, in the development of increasingly bigger and more complex systems the effects, problems and opportunities of the system's overall structure, organisation and emergent behaviour. In a broad definition, the architecture of a system describes its fundamental organisation, which illuminates the top level design decisions, namely

- how is it composed and of which interacting parts?
- which are the interactions and communication patterns present?
- which are the key properties of parts the overall system rely and/or enforce?

As a model it acts as an abstraction of a system that surpresses details of elements that do not affect how they use, are used by, relate to or interact with other elements. Therefore it focus on the structural elements and their interfaces by which a system is composed, their separate and joint behaviour as specified in collaborations among those elements, and finally the composition of these structural and behavioral elements into larger subsystems. According to norm ANSI/IEEE Std 1471-2000, which is part of a on-going standardisation effort, it describes *the fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.*

Reactive systems. Software Architecture, as a design discipline, is currently challenged by the continuous evolution towards very large, heterogeneous, highly dynamic computing systems, which require innovative approaches to master their complexity. Actually, current software systems are large and inherently complex. The behaviour they exhibit cannot be simply characterised in terms of a relation between input and output data values. In most cases, such a behaviour

- is potentially non-terminating,
- expresses a continued interaction with the system's environment and sub-systems which execute concurrently in distributed, often loosely coupled configurations.

In general we refer to this sort of systems as *reactive*, adopting a term coined by Amir Pnueli and David Harel [HP85] in the 80's.

Designing such systems right is very difficult, because it involves not only mastering the complexity of building and deploying a large application in time and budget, but also managing an open-ended structure of autonomous components, typically distributed and highly heterogeneous. Additional difficulties arise with the need to take into account a plethora of issues such

as real-time responsiveness, dynamic reconfiguration, QoS-awareness, self-adaptability, security, dependability, under-specification of third-party components, among many others.

The course. There is no general-purpose, universally tailored approach to the architectural design of reactive systems. To face a complex, heterogeneous reality, it is probably prudent to concentrate on specific classes of these systems and study, for each of them, the relevant approaches, methods and tools.

On the other hand, despite remarkable progress in the representation and use of software architecture, specification of architectural designs remain, at present, largely informal. Typically, it relies on graphical notations with poor semantics, and often limited to express only the most basic structural properties. In such a context, this course clearly deviates from more common, or classical, introductions to software architecture by simultaneously

- avoiding restriction to a unique framework or methodology, focussing instead on three sorts of formal structures (*processes*, *connectors* and *Mealy machines*) which can be taken as a basis for architectural reasoning;
- strengthening its effectiveness by focussing exclusively on models amenable to formal verification and reasoning, according to the rationale that only a *formal* perspective has potential to be effective in framing architectural design in sound Engineering standards.

Finally, the course introduces a specific section on *behavioural restrictions in software architecture*, dealing with *real-time* and *stochastic* behaviour of different architectural entities. The former is useful in a number of concrete situations, namely for the design of time-critical systems. Stochastic models, on the other hand, are essential to deal with unreliable behaviour (quantifying non determinism), as well as to forecast system's performance and evaluating quantitative properties (*e.g.*, queue length, waiting time, etc). Actually, in many, modern reactive systems the difference between their functional features and their performance properties blurred as relevant functionalities become inextricably linked to performance aspects.

2 Syllabus

1. **Introduction to software architecture**
2. **Background: Transition structures and modal logics**
   - transition structures
   - simulation and bisimulation
   - modal logics
3. **Process-oriented architectural design**
   - processes and process algebra for architectural description
   - brief introduction of MCRL2: review of process algebra and logics
   - a process-oriented ADL: ARCHERY
4. **Coordination-oriented architectural design**
   - Exogenous coordination for architectural description
   - REO: principles, models, examples
   - REO semantics: constraint automata & double streams
   - Architectural properties and their verification (VEREOFY)
5. **Component-oriented architectural design**
   - components as *monadic Mealy machines*
   - a component calculus
   - MMM: an HASKELL library
6. **Behavioural constraints**
   - *Real-time* constrains: timed automata and UPPAAL
   - *Stochastic* constrains: Markov chains, PRISM and STOCHASTIC REO

Note. The course consists of a series of formal lectures interspersed with exercise and laboratorial classes under tutorial supervision. In the latter, students are encouraged to work in groups and discuss the course material. Bibliographic references, slides handouts and exercise suggestions will be given with the summary of each lecture. Assessment is continuous, taking into considerations the resolution of proposed exercises (up to 30%) and a individual project (up to 70%).

3 Summaries

Lecture 1 - 2011.11.24, 09:00-13:00

Summary. Software architecture: concepts, objectives, methodologies. Architectures for reactive systems. Introduction to MFES architectural module: aims, syllabus, resources.

Resources

Lecture notes: **L1PAS-11-12.pdf** (available from MFES website)

References:

- To read: [Gar04] (available from MFES website)
- A classical perspective on architectural discipline: [Gor06]
  (table of contents and chapter 1 available from MFES website)
- Other: [AG97, GS93, vL04, AF04] (available on request)

Exercises Read [Gar04] and chapter 1 of [Gor06] in confront to the lecture. Make a brief summary and discussion.

Lecture 2 - 2011.11.24, 14:00-18:00

Summary. Background: transition structures and labelled transition systems. Morphism. Simulation and bisimulation: basic definitions and properties.

Resources

Lecture notes: **L2PAS-11-12.pdf** (available from MFES website)

Exercises

1. Given two transition systems $\langle S_1, T_1 \rangle$ e $\langle S_2, T_2 \rangle$ over an alphabet A, two states p and q are *mutually similar* iff
$$p \doteqdot q \;\Leftrightarrow\; p \lesssim q \wedge q \lesssim p$$
Prove that $\doteqdot$ is an equivalence relation. Compare it with bisimilarity.

2. Consider the following transition relation:
$$\begin{aligned} T \;= \{&\langle 1,a,2\rangle, \langle 1,a,3\rangle, \langle 2,a,3\rangle, \langle 2,b,1\rangle, \langle 3,a,3\rangle, \langle 3,b,1\rangle, \langle 4,a,5\rangle, \langle 5,a,5\rangle, \\ &\langle 5,b,6\rangle, \langle 6,a,5\rangle, \langle 7,a,8\rangle, \langle 8,a,8\rangle, \langle 8,b,7\rangle\} \end{aligned}$$
Prove or refute that $1 \sim 4 \sim 6 \sim 7$.

3. Show that the set of all bisimulations between two given transition systems is a complete lattice, ordered by inclusion. What is the top of this lattice?

4. A *trace* in a transition system is a sequence of labels such that there is a sequence of states $s_0, s_1, ...$ such that
$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$$

with $s = a_1 a_2 a_3 \cdots a_n$. A trace is *complete* if it leads to a state with no further transitions. Prove that two bisimilar states have the same set of traces. Would this result remain valid if complete traces were considered instead?

Lecture 3 - 2011.12.15, 09:00-13:00

Summary. Process-oriented architectural design: introduction to process algebra and behaviour modelling.

Resources

Lecture notes: **L3PAS-11-12.pdf** (available from MFES website)

References:

- Process algebra tool: mCRL2 (available from www.mcrl2.org/)
- References on **process algebra**: [BPS01, BBR09, Fok07, Mil99]

Exercises

1. Install the mCRL2 and explore the examples provided.
2. Consider two popular *architectural styles — client-server* and *balckboard*. Model these styles in mCRL2 and devise their application in an example.

Lecture 4 - 2011.12.15, 14:00-18:00

Summary. Process-oriented architectural design: the Archery experimental ADL.

Resources

Lecture notes: talk by Alejandro Sanchez

References:

- The Archery tool: Archery
  (available from `www.unsl.edu.ar/~asanchez/index.php?page=archery`)
- Recent book on **process algebra as a basis for ADL**: [ABC10]

Exercises

1. Install Archery and specify two applications of popular architectural styles — *client-server* and *blackboard*.

Lectures 5 and 6 - 2012.01.12, 09:00-13:00 and 14:00-18:00

Summary. Process-oriented architectural design: expressing and verifying requirements. Introduction to modal logic (modalities, semantics, expressivity, richer logics). Hybrid logic. Hennessy-Milner logic and the modal μ-calculus.

Resources

Lecture notes: **L4PAS-11-12.pdf** (available from MFES website)

References:

- A well-written textbook on **modal logic** : [BdRV01]
  (table of contents and chapter 1 available from MFES website)
- An introduction to Hennessy-Milner logic and the μ-calculus: [Sti96]

Project (part 1)

- Choose a problem from the list of small case studies published in the MFES website. Design an architectural solution for it, either in Archery or directly in mCRL2. Formulate modal properties for this problem and verify them against your architectural specification using the tools provided by mCRL2.

Lecture 7 - 2012.01.19, 14:00-18:00

Summary. Process-oriented architectural design: examples and exercises. Experiments with mCRL2.

Resources

Lecture notes: **PAS-TPSession1.pdf** (available from MFES website)

References:

- A case study from a book in preparation by Groote Reniers (2010) (available from MFES website)

Lecture 8 - 2012.01.26, 09:00-13:00

Summary. Coordination-oriented architectural design. Introduction to Reo.

Resources

Lecture notes: **L5PAS-11-12.pdf** (available from MFES website)

References on Reo (original papers): ([Arb03, Arb04] and F. Arbab Inaugural Lecture at Leiden (available from MFES website)

Lecture 9 - 2012.01.26, 14:00-18:00

Summary. Coordination-oriented architectural design: examples and exercises. Experiments with ReoTools. The colouring semantics for Reo connectors.

Resources

Lecture notes: **PAS-TPSession2.pdf** (available from MFES website, by Nuno Oliveira)

References:

- ReoTools (available from reo.project.cwi.nl/)

Project (part 2)

- Choose a problem from the list of small case studies published in the MFES website. Design an architectural solution in Reo and animate it with ReoTools. As a bonus you may like to explore Vereofy model checker in your problem.

Lecture 10 - 2012.02.09, 09:00-13:00

Summary. Coordination-oriented architectural design: formal semantics for Reo. Expressing architectural patterns.

Resources

Lecture notes: **L5PAS-11-12.pdf** (available from MFES website)

References on Reo (semantic models): [BSAR06, CCA07, AR03] (the first two are available from MFES website)

References

[ABC10] A. Aldini, M. Bernardo, and F. Corradini. *A Process Algebraic Approach to Software Architecture*. Springer Verlag, 2010.

[AF04] L. Andrade and J. L.. Fiadeiro. Architecture based evolution of software systems. In M. Bernardo and P. Inverardi, editors, *Third International Summer School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures (SFM 2003)*, pages 148–181. Springer Lect. Notes Comp. Sci, Tutorial, (2004), Bertinoro, Italy, September 2004.

[AG97] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM TOSEM*, 6(3):213–249, 1997.

[AR03] F. Arbab and J. J. M. M. Rutten. A coinductive calculus of component connectors. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Recent Trends in Algebraic Development Techniques, 16th Inter. Workshop, WADT 2002, Revised Selected Papers*, pages 34–55. Springer Lect. Notes Comp. Sci. (2755), 2003.

[Arb03] F. Arbab. Abstract behaviour types: a foundation model for components and their composition. In F. S. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Proc. First International Symposium on Formal Methods for Components and Objects (FMCO'02)*, pages 33–70. Springer Lect. Notes Comp. Sci. (2852), 2003.

[Arb04] F. Arbab. Reo: a channel–based coordination model for component composition. *Mathematical Structures in Comp. Sci.*, 14(3):329–366, 2004.

[BBR09] J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2009.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Scie*. Cambridge University Press, Cambridge, 2001.

[BPS01] J. Bergstra, A. Ponse, and S. Smolka. *Handbook of Process Algebra*. Elsevier, 2001.

[BSAR06] C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *Science of Computer Programming*, 61(2):75–113, 2006.

[CCA07] D. Clarke, D. Costa, and F. Arbab. Connector colouring I: Synchronisation and context dependency. *Science of Computer Programming*, 66(3):205–225, 2007.

[Fok07] W. Fokkink. *Modelling Distributed Systems*. Texts in Theoretical Computer Science. Springer Verlag, 2007.

[Gar04] D. Garlan. Formal modeling and analysis of software architecture: Components, connectors and events. In M. Bernardo and P. Inverardi, editors, *Third International Summer School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures (SFM 2003)*, pages 1–24. Springer Lect. Notes Comp. Sci, Tutorial, (2004), Bertinoro, Italy, September 2004.

[Gor06] Ian Gorton. *Essential Software Architecture*. Springer Verlag, 2006.

[GS93] 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.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume 13 of *NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci.*, pages 477–498. Springer-Verlag, 1985.

[Mil99] R. Milner. *Communicating and Mobile Processes: the π-Calculus*. Cambridge University Press, 1999.

[Sti96] Colin Stirling. Modal and temporal logics for processes. In *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, August 27 - September 3, 1995, Proceedings)*, volume 1043 of *Lecture Notes in Computer Science*, pages 149–237. Springer, 1996.

[vL04] Axel van Lamsweerde. From system goals to software architecture. In M. Bernardo and P. Inverardi, editors, *Third International Summer School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures (SFM 2003)*, pages 25–43. Springer Lect. Notes Comp. Sci, Tutorial, (2004), Bertinoro, Italy, September 2004.