Software architecture for reactive systems (introduction)

Luís Soares Barbosa

HASLab - INESC TEC
Universidade do Minho
Braga, Portugal

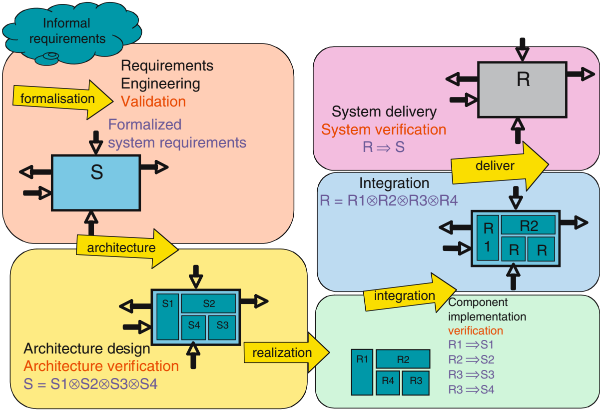
February 2019

Software Engineering

Software development as one of the most complex but at the same time most effective tasks in the engineering of innovative applications:

- Software drives innovation in many application domains
- Appropriate software provides engineering solutions that can calculate results, communicate messages, control devices, animate and reason about all kinds of information
- Actually software is becoming everyware ...

Software Engineering



(illustration from [Broy, 2007])

Software Engineering

So, ... yet another module in the MFES profile?

Software architecture for reactive systems

characterised by

- a methodological shift: an architectural perspective
- a focus: on reactive systems
- this year with a major extension to quantum systems

What is software architecture?

[Garlan & Shaw, 1993]

the systematic study of the overall structure of software systems

[Perry & Wolf, 1992]

SA = { Elements (*what*), Form (*how*), Rationale (*why*) }

[Kruchten, 1995]

deals with the design and implementation of the high-level structure of software

[Britton, 2000]

a discipline of generic design

What is software architecture?

[Garlan & Perry, 1995]

the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time

[ANSI/IEEE Std 1471-2000]

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.

[Garlan, 2003]

a bridge between requirements and code (...) a blueprint for implementation.

What is software architecture?

The architecture of a system describes its gross structure which illuminates the top level design decisions, namely

- how is it composed and of which interacting parts?
- where are the pathways of interaction?
- which are the key properties of the parts the architecture rely and/or enforce?

Two examples

from the micro level (a Unix shell script)

```
cat invoices | grep january | sort
```

- Application architecture can be understood based on very few rules
- Applications can be composed by non-programmers
- ... a simple architectural concept that can be comprehended and applied by a broad audience

Two examples

to the macro level (the WWW architecture)

- Architecture is totally separated from the code
- There is no single piece of code that implements the architecture
- There are multiple pieces of code that implement the various components of the architecture (e.g., different browsers)
- One of the most successful applications is only understood adequately from an architectural point of view

Reactive systems

Reactive system

system that computes by reacting to stimuli from its environment along its overall computation

- in contrast to sequential systems whose meaning is defined by the results of finite computations, the behaviour of reactive systems is mainly determined by interaction and mobility of non-terminating processes, evolving concurrently.
- observation $\equiv$ interaction
- behaviour $\equiv$ a structured record of interactions

Reactive systems

Concurrency vs interaction

$$x := 0;$$
$$x := x + 1 \mid x := x + 2$$

- both statements in parallel could read x before it is written
- which values can x take?
- which is the program outcome if exclusive access to memory and atomic execution of assignments is guaranteed?

Challenges

Software architecture for reactive systems

- new target: need for an architectural discipline for reactive systems
  (often complex, time critical, mobile, cyber-physical, etc ...)
- from composition to coordination (orchestration)
- relevance of wrappers and component adapters: integration *vs* incompatible assumptions about component interaction
- reconfigurability
- continued interaction as a first-class citizen and the main form of software composition

Our approach

There is no general-purpose, universally tailored, approach to architectural design of complex and reactive systems

Therefore, the course

- introduces different models for reactive systems
- discusses their architectural design and analysis
- with (reasonable) tool support for modelling and analysis

But why bringing quantum into the picture?

- Computer Science and Information theory progressed by abstracting from the physical reality.
- ... this was the key of its success to an extent that its origin was almost forgotten
- On the other hand quantum mechanics ubiquitously underlies ICT devices and the implementation level (e.g. transistor, laser, ...),
- but had no influence on the computational model itself
- ... until now when two main intelectual achievements of the 20th century met — Computer Science and Quantum Mechanics — and quantum effects are used as computational resources

But why bringing quantum into the picture?

The second quantum revolution

For the first time the viability of quantum computing may be demonstrated in a number of real problems extremely difficult to handle, if possible at all, classically, and its utility discussed across industries.

- huge investment by both the States, large companies and startups
- the race for quantum rising between major IT players (e.g. IBM, Intel, Google, Microsoft)
- proof-of-concept machines up to 50 qubits until the end of 2018
- national and regional programmes (from the 2016 Quantum Manifesto to the EU QT Flagship and this week announcement of FCT Call for PhD grants)

Invitation to a fast running train ...

Academic IBM Q HUB since September, 1, 2018

- Part of the worldwide IBM Q Network of companies and academies to exploit potential applications of Quantum Computing in Industry
- Real time, full access to new quantum machines
- Multidisciplinar, dedicated teams
- A problem-driven research
- International cooperation

Syllabus

- Software architecture, processes and interaction
- Classical reactive processes
  - (Modelling) Introduction to transition systems and process algebra
  - (Verification) Introduction to modal, hybrid and dynamic logic
  - (Tool) The mCRL2 framework
  - Variants: (Timed | Probabilistic | Hybrid) processes
- Quantum processes
  - (Modelling) The quantum computational model
  - (Modelling) Quantum algorithmic processes
  - (Verification) Dynamic logic for quantum processes
  - (Tool) The Qiskit platform
- Coordination-oriented architectures
  - The Reo exogenous coordination model
  - Compositional specification of the glue layer

Pragmatics

`http://arca.di.uminho.pt/ac-1819/`

Special events ...

21 Feb	:	all-day lecture (replacing 28 Feb)
11 Apr	:	all-day workshop Quantum Days (replacing 4 Apr)
23 May	:	all-day short crash course on Reo (by F. Arbab, CWI) (replacing 9 May)

Pragmatics ...

- **Assessment:**
  - Test in June - 70 %
  - Group projects (2x) - 40 % (10+20)

http://arca.di.uminho.pt/ac-1718

- **Research context:** Projects
  - DALI — 2016-18
    on Dynamic logics for cyber-physical systems
  - TRUST — 2016-18
    on Trustworthy Software Design with Alloy

possible GRANTS available!

(with INL, U. Aveiro, CWI, INESC TEC)

Model checking

Recall "Especificação e Modelação":

- Modelling reactive systems – Kripke structures and NuSMV
- Specification – Temporal logics (LTL ~~and CTL/CTL*~~)
- Verification – Check if a formula holds in a system

SMV model checker

What we will see

- Labelled transition systems (LTS) as Kripke structures
  - Process algebra (not ~~Petri Nets~~ SMV) to define LTS
  - mCRL2 toolset to model (not SMV)
  - Equivalence of LTS
- Modal logics – generalising temporal logics (CTL*,CTL,LTL)
- Using mCRL2 toolset to verify properties

- Later: Timed-automata and UPPAAL model checker (CTL)

Model

$\mathfrak{M}, w \models \phi$ – what does it mean?

Model definition

A model for the language is a pair $\mathfrak{M} = \langle \mathfrak{F}, V \rangle$, where

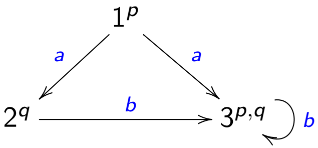
- $\mathfrak{F} = \langle W, \{R_m\}_{m \in \text{MOD}} \rangle$
  is a Kripke frame, ie, a non empty set W and a family R_m of binary relations (called *accessibility relations*) over W, one for each modality symbol $m \in \text{MOD}$. Elements of W are called points, states, worlds or simply vertices in directed graphs.
- $V : \text{PROP} \longrightarrow \mathcal{P}(W)$ is a valuation.

Kripke structures from last semester

- $\text{MOD} = \{\mathbf{1}\}$
- (S, I, R, L) where $S = W$, $I = \{w\}$, $R = R_{\mathbf{1}}$, $L = V$
- $\mathfrak{F} = \langle W, R \rangle$ instead of $\mathfrak{F} = \langle W, \{R_m\}_{m \in \text{MOD}} \rangle$

Example

$\mathfrak{M} =$

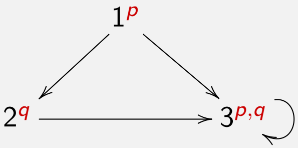


$$
\begin{aligned}
W &= \{1, 2, 3\} \\
MOD &= \{a, b\} \\
R_a &= \{(1,2), (1,3)\} \\
R_b &= \{(2,3), (3,3)\} \\
V &= \{1 \mapsto \{p\}, \\
&\quad\; 2 \mapsto \{q\}, \\
&\quad\; 3 \mapsto \{p, q\}\}
\end{aligned}
$$

- $\mathfrak{M}, 1 \models p$
  means p holds in state 1
- $\mathfrak{M}, 2 \models [b]\, p$
  means p holds in every state reachable with b from 2.

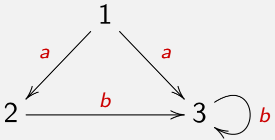
Key differences

Before



- emphasize on states - desired/forbidden states
- SMV language to generate models
- $\mathfrak{M}, 1 \models p$, $\mathfrak{M}, 1 \models F\ G\ p$

Now



- emphasize on actions - desired/forbidden sequences
- Process algebra to generate models
- $\mathfrak{M}, 2 \models [a]\,\text{false}$