Whither Software Architecture?

Jeff Kramer
Imperial College London

“... software architecture is a set of architectural (or, if you will, design) elements that have a particular form.” (Perry, Wolf 1992)

“The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.” (SEI 2010)

“A software system’s architecture is the set of principal design decisions made during its development and any subsequent evolution.” (Taylor, Medvidovic, Dashofy 2010)

Whither software architecture

- how did we get here?
- impact?
- where are we going?

a “soap opera” based on my personal research experience

unintentional stepping on toes
**my formative project**

CONIC - “configuration programming”

**the CONIC project**

Computer Control & Monitoring of underground systems in coal mining.

The investigators: Guess Who and Morris Sloman

The research assistant: Jeff Magee

**coal mines**

Underground, coal mines consist of a number of interacting subsystems:

✦ coal cutting
✦ transport
✦ ventilation
✦ drainage
✦ ...

... changes as the mine topography changes.

**requirements elicitation**

➡ complex
large number of interconnected devices, sensors, actuators, controllers

➡ highly distributed
over the mine site, both above and below ground

➡ evolving
new coal faces open, old faces close

➡ robust
against failures

Software structure should mirror the physical mine...
1. distributable components

Key property of **context independence**
• communication via a well-defined interface.
• supports third party instantiation and binding
• reuse

- input and output ports (indirection)
- parameterised component types

2. interaction

- unidirectional
- asynchronous
- bidirectional
- rendezvous

transient
local/remote
communication.

3. configuration (static)

Separate explicit description of the **structure** of the system in terms of the **composition** of component instances and connections (i.e. third party instantiation and binding).

Hierarchical composition helps to hide complexity.
3. configuration (... & dynamic)

- Reusable components
  The control software for a particular coal mine could be assembled from a set of components.

- On-line change
  Once installed, the software could be dynamically modified without stopping the entire system to deal with new coalfaces.

Research team:
Kevin Twidle
Naranker Dulay
Keng Ng

CONIC was not general

- ... was programming language dependent (Pascal)
- ... had fixed communications primitives
- ... had simple single message interfaces for bindings

Structural view provides a useful level of abstraction.
**Component types** have one or more interfaces. An **interface** is simply a set of names provided or required by the component, referring to actions in a specification or services in an implementation.

**System types** are composed hierarchically by component instantiation and interface binding.

**Tool support** graphical design and software system generation, deployment, as well as ...

**Process Calculus - FSP**

- Component behaviour:
  
  - PUMP = STOPPED,
  - STOPPED = (cmd.start -> STARTED),
  - STARTED = (pump -> STARTED | cmd.stop -> STOPPED).

- Model architecture:
  
  \[
  
  \|\|\text{PUMP\_CONTROL} = (c:\text{CONTROL} \| p:\text{PUMP}) \\
  /\{c.cmd/p.cmd, \\
  \text{level}/c.level, \\
  \text{pump}/p.pump\}.
  
  \]

**Analysis - LTSA**

- Fluent:
  
  - RUNNING = <start, stop>
  - METHANE = <methane.high, methane.low>

- Assert:
  
  - SAFE = [\!(\text{tick} \rightarrow \text{METHANE} \rightarrow \neg \text{RUNNING})]

ESEC/FSE 1995, FSE 1996

ESEC/FSE 1995, FSE 1996

ESEC/FSE 2005
... in collaboration as always ...

Jeff Magee

Shing-Chi Cheung
- LTS, CRA & Safety

Nat Pryce
- Animation

Dimitra Giannakopoulou
- Liveness & Fluent LTL

Emmanuel Letier
- AFLTL

Sebastian Uchitel
- Synthesis


connector wars

pragmatists Vs purists?

connector wars

Component
Connector
Component

pragmatists Vs purists?

Component
Connector
Component

Component
"Connector"
Component

connector wars

pragmatists Vs purists?
Impact?

Koala

In the ARES project, Rob van Ommering saw potential of Darwin in specifying television product architectures and developed Koala, based on Darwin, for Philips.

First large-scale industrial application of an ADL.

Koala - example

Success.

... and is still in use.

But ...
Koala

Not more widely adopted, even in Philips!

• ... despite right level of abstraction
• ... despite compiler + code generation
• ... despite support for diversity

WHY???

Is Koala the only ADL in use?

ROOM
MetaH
AADL
UNICON
WRIGHT
ACME
Rapide
C2
xADL
ArchJava
SADL
UML2?
...

ADLs have not been widely adopted!

Disappointed but not downhearted...

“Not invented elsewhere!”

“All hat and no cattle!”
Architecture research is a success

The abstractions pioneered in software architecture research have actually been very influential.

- qualitative aspects
- reviews/style guides
- architectural patterns
- provides and requires
- UML2
- modelling and analysis

Garlan and Shaw (ESEC/FSE 2011)

Why were ADLs not widely adopted?

Object-Oriented Programming became mainstream:
- focus on class hierarchy
- implicit requires interfaces
- objects rather than components

components vs objects

- benefits of a component oriented view are recognised
- we can gain the benefits even with objects.

components from objects

- component type as an OO class
- dependency injection (or inversion of control):
  - “new” and connections are no longer in the component code
  - supports 3rd party instantiation and binding
components from objects

```java
public class Leaf {
    public int attribute = 5;
    private Interface port1 = new Interface();
    { ...Interface methods ...; }
    public Interface getPort1() {
        return port1();
    }
    private Interface port2;
    public void setPort2(Interface i) {
        port2 = i;
    }
}
```

```java
public class Composite {
    private Leaf a = new Leaf();
    private Leaf b = new Leaf();
    public Composite() {
        a.setPort2(b.getPort1());
    }
    public Interface getPortA() {
        return a.getPort1();
    }
    public void setPortB(Interface i) {
        b.setPort2(i);}
```

composite components

dependency injection

Permits separation of configuration from use

- current EJB (CDI) - “… server-side component architecture for Java”
- Spring - “… application development framework for enterprise Java”
- Guice - “… lightweight dependency injection framework for Java 5 and above”
- Autofac - “… IOC container for .NET classes by treating them as components.

rays of hope for ADLs

- some current practice in programming languages and some application domains
- research on change:
  1. software maintenance and evolution
  2. adaptive software
1. ADLs for software evolution

Change is intrinsic in the architecture definition

- add three basic constructs to a Darwin-like ADL (Backbone) to support arbitrary extension:
  - resemblance, replacement, strata

- Evolve Tool uses UML2 graphical notation

SAVCBS 2006, ICSE 2011

---

**resemblance**

```
A          resembles
\[ \begin{array}{c}
\quad \text{Shape} \\
\quad \text{denotes UUID}
\end{array} \]
```

define new components as a delta from the structure of one or more existing components (i.e., reuse)

---

**replacement**

```
A          replaces
A'\[ \begin{array}{c}
\quad \text{A' globally replaces A in the architecture.}
\end{array} \]
```

---

**evolution**

```
A          evolves
A'\[ \begin{array}{c}
\quad \text{combines resemblance and replacement}
\end{array} \]
```

---

Andrew McVeigh

Jeff Magee
stratum

- packages the definitions
- unit of ownership
- controls visibility

extension

depends

base

include both strata to give extended system

include base stratum to give original system

decentralised development

Used by U

+ conflict detection

Extended by X

Base’ +

Extended by Y

Base” +

Base

Developed by D

incremental extension properties

- **ALTER**
  - Allows any possible extension even if unplanned

- **NO IMPACT**
  - Others are not impacted by extensions they don’t want

- **DECENTRALIZED**
  - Supports a fully decentralized environment

- **COMBINE**
  - Extensions / upgrades can be combined, problems rectified

- **NO SOURCE**
  - Works even without source code!

Evolve demo

- Evolve design tool
- Backbone ADL

ICSE demo 2011

http://www.intrinsarc.com/evolve
conformance

- “What are the prospects for showing conformance between architecture and code?”

  question posed by Garlan and Shaw (ESEC/FSE 2011)

Generate it!

... and store it in the code (“exoskeletal software”)

2. ADLs for adaptive software

from change in the form of

- maintenance and evolution
to

- self-managed software adaptation

three layer architecture model

- a single feedback loop?
- response times?
- complexity?

MAPE cycle

Plan synthesis based on a domain model and goals

Decentralised component selection and assembly by transitive closure on components satisfying plan actions

Safe operation, including during change (tranquility)

a separation of timescales and concerns

ICSE FOSE ’07, SEAMS 2008, SEAMS 2011
generating the architecture

generating revised plans

In conclusion...

Previously on "Whither Software Architecture"
**Architecture as an Abstraction**

... the same architectural description can be used as the structural framework to hang requirements, to compose behaviours for analysis, to compose component implementations for systems, ....

**continuing research...**

- partial component model synthesis from goals and scenarios for architectural fragments,
  - merge overlapping models,
  - compose component models according to the system architecture

- requirements elaboration and revision using a combination of model checking and machine learning


**a life of research**

interesting, challenging, rewarding, fun

**acknowledgement**

Sebastian Uchitel
Dalal Alrajeh
Alessandra Russo
Axel van Lamsweerde
Whither
Software Architecture?

Jeff Kramer
Imperial College London