A Ten-Year Perspective
on
“Formalizing Architectural Connection”

David Garlan
Carnegie Mellon University
Robert Allen
IBM

ICSE 2004

---

Figure 3.1 The Configuration of the GENESIS Prototype

Legend:
- Module or program
- Module or table
- A calls B
- Data path

David Garlan
May 2004
ICSE 2004 Most Influential Paper Presentation

Figure 3.1 DBMS and Its Environment


David Garlan
May 2004
ICSE 2004 Most Influential Paper Presentation

Figure 5. Framework for a single-enterprise Web-service-enabled application that disseminates secure documents. The numbered arrows show the steps in the authorization process.
Issues Addressed by an Architectural Design

- **Gross decomposition of a system into interacting components**
  - typically hierarchical
  - using rich abstractions for component interaction
    (or system “glue”)
  - often using common design idioms/styles

- **Emergent system properties**
  - performance, throughput, latencies
  - reliability, security, fault tolerance, evolvability

- **Rationale and assignment of function to components**
  - relates requirements and implementations

- **Envelope of allowed change**
  - “load-bearing walls”, limits of scalability and adaptation
  - design idioms and styles

The Challenge in 1994

- **How can we establish intellectual control over this new world?**
  1. express arch descriptions precisely and intuitively
    - both syntax and semantics
  2. develop soundness criteria & tools to check them
    - what type checkers and linkers do for current systems
  3. analyze architectures to determine key properties
    - such as performance, reliability, change impact, interoperability
  4. exploit patterns and styles
    - without unnecessary proliferation and isolation of new design languages and tools
Important Precedents

- **Module Decomposition**
  - Criteria for module decomposition (e.g., Inf. Hiding)

- **Module Interconnection Languages**
  - Separating system structure from implementation details

- **Identification of software architecture**
  - Perry & Wolf, Kramer & Magee, Garlan & Shaw
  - Mary Shaw: codification; connectors should be “first class”

- **Product line successes**
  - e.g., Tektronix

Why We Wrote This Paper

- A crucial step in understanding an architecture is to know what the lines (connectors) mean

- Existing work on system structure allowed the expression of only simple interactions
  - e.g., procedure calls, data accesses

- In real architectural descriptions connectors often represent complex multi-way interactions

- Understanding whether a set of components can legally interact only makes sense if you know what the interactions are
Typical questions we wanted to be able to answer

- Which is the reading/writing end of the pipe?
- Is writing synchronous?
- What if F₂ tries to read and the pipe is empty?
- Can F₁ choose to stop writing?
- Can F₂ choose to stop reading without consuming all of the data?
- If F₁ closes the pipe, can it start writing again?
- If F₂ never reads, can F₁ write indefinitely?

What the paper said

- Think of connectors as representing protocols
  - define abstract behavior of interaction
- Decompose that protocol into parts representing
  - each of the participating parties (roles)
  - a mediating mechanism (glue)
- Define checkable properties
  - Consistency of the connector
  - Compatibility between a role and a component interface (port -- also represented as a protocol)
Specifying Connector Behavior

Wright: a variant of CSP (Hoare 85)

- Events: $e, \text{request, read?y, write!5}$
- Processes: $P, \text{Reader, Writer, Client, §}$
  - Sequence: $e \rightarrow P, \hspace{2mm} P ; Q$
  - Choice: $P \sqcap Q, \hspace{2mm} P \sqcup Q$
  - Composition: $P \parallel Q$

A Pipe Connector

Connector $\text{Pipe}$

Role $\text{Writer} = (\text{write!x} \rightarrow \text{Writer}) \sqcap (\text{close} \rightarrow §)$

Role $\text{Reader} = \text{Read} \sqcap \text{Exit}$
  where $\text{Read} = (\text{read?x} \rightarrow \text{Reader}) [] (\text{eof} \rightarrow \text{Exit})$
  $\text{Exit} = \text{close} \rightarrow §$

Glue = $\text{Writer}.\text{write?x} \rightarrow \text{Glue} []$
  $\text{Reader}.\text{read!y} \rightarrow \text{Glue} []$
  $\text{Writer}.\text{close} \rightarrow \text{ReadOnly} []$
  $\text{Reader}.\text{close} \rightarrow \text{WriteOnly}$

where ...
So What?

- **We have a semantics for architectural connectors**
  - E.g., allows us to answer all of those questions about pipes

- **We can determine whether the pipe is consistent**
  - Roughly: Reader || Writer || Glue does not deadlock

- **We can check whether a filter’s interface is compatible with a pipe role**
  - Roughly: F.out refines Pipe.Writer

- **We can use tools to do this checking automatically**
  - We used FDR, a model checker for CSP

What Happened After

- **Applications to non-trivial systems**
  - HLA specification identified over 80 issues, some serious

- **Other ways of formalizing connectors**
  - Different protocol languages
  - Other properties

- **Incorporation of first class connectors in architecture design languages and tools**
  - C2, Acme, SADL, ADML, xADL, ..

- **Influences on standards**
  - UML 2.0 now has a new notion of “port” and “connector”
  - J2EE, JMX, etc.
What we got right

- Software Architecture is not programming
  - new concepts are needed to express it
- SA is about run-time structures and properties
  - locus of reasoning about performance, reliability, etc.
- Connectors are where the action is
  - get this right and the rest is gravy
- Connectors must be first-class
  - specify connector types, create on the fly, transform, …
- Architectural mismatch is a core problem
  - and many mismatches are interaction mismatches
- Architectural behavior is worth specifying
  - reusing existing formalisms for behavior
What we got wrong

- Architectures are largely static
- Software architecture is a top-down activity
- Functional and extra-functional properties are orthogonal
- Mapping architecture to code is just a version of the well-understood refinement problem

It's just a ween o'blethers

A Few Open Questions

- How can we specify and reason about dynamic architectures?
- How can we relate architectures and code?
- Can we create effective connector calculi?
- How can we exploit architectures at run time?
- What are practical methods of refinement for architecture?

Let the tow gang wi’ the bucket
Conclusion

- Software architecture emerged in the 1990s as a significant new area of study
- First-class connectors played a central role in establishing its direction
- Behavioral specifications of connectors provides substantial analytic leverage
- There remain many interesting and relevant research problems

Moral: to the vector belong the spoils

Scottish Glossary

- Cauld kail het again
  - Cold kale heated up (= reuse)
- It’s just a ween o’blethers
  - It’s just a pack of rubbish
- Let the tow gang wi’ the bucket
  - Let things take their course
- Bide awee
  - Wait a bit