ECE750-Topic11: Component-Based Software

Software Architecture and Components

Ladan Tahvildari
Assistant Professor
Dept. of Elect. & Comp. Eng.
University of Waterloo
Paradigm Shifts in Software

SA and Components

- The Role of Software Architecture
- Designing Software Architectures
- Architecture-driven Component Development
- Component-driven Architecture Development
The Software Architecture

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software components [and connectors], the externally visible properties of those components [and connectors] and the relationships among them.”
Role of the Software Architecture

- The main uses of a software architecture are:
  - Assessment and evaluation
  - Configuration management
  - Dynamic software architectures
Assessment and Evaluation

- **Stakeholder-Based Assessment**
  - is concerned with determining whether the trade-offs between requirements in the software architecture match the actual stakeholder priorities of these requirements.
    - SAAM (Software Architecture Analysis Method)
    - ATAM (Architecture Tradeoff Analysis Method)

- **Quality-Attribute Oriented Assessment**
  - aims at providing a quantitative prediction of one quality attribute (e.g. maintainability, performance, reliability or security)
    - QDR (Quality-Driven Re-engineering) Framework
Configuration Management

- The software architecture is frequently used as a means to manage the configuration of the product.
Dynamic Software Architectures

- The software architecture should *reorganize itself in response to the dynamic change* of the systems' quality requirements.

- Maintained even during run-time.
Can be seen as a function that:

- Takes a requirement specification as input.
- Generates an architectural design as output.
- Is not an automated process, necessitating great effort and creativity from the involved software architects.

Is comprised of three steps:

- Functionality-Based Design
- Assessment of the Quality Attributes
- Architecture Transformation
Software Architecture

Design Process

- Requirement specification
- Requirement selection
  - (Partial) requirement specification
  - F.R.
  - (Partial) requirement specification
  - OK
  - Yes
- Functionality-based architectural design
  - Application architecture
  - Architecture transformation
  - QA-optimizing solutions
  - not OK
  - Estimate quality attributes
    - OK
    - More Requirements?
      - no
Functionality-Based Design

- The design process starts with functionality-based design and consists of four steps:
  - Defining the **boundaries and context** of the system.
  - Identification of **archetypes**.
  - **Decomposition of the system** into its main components.
  - The first **validation of the architecture** by describing a number of system instances.
Assessment of the Quality Attributes

- The second phase is the assessment of the quality attributes in which:
  - Each quality attribute is given *an estimate*
    - If all estimated quality attributes are as good or better than required, the architectural design process is finished
    - If not the third phase of software architecture design is entered: architecture transformation
Architecture Transformation

- Is concerned with selecting design solutions to improve the quality attributes while preserving the domain functionality:
  
  - The design is again evaluated and the same process is repeated if necessary.
  
  - The transformations (i.e. quality attribute optimizing solutions) generally improve one or some quality attributes while they affect others negatively.
Architecture Transformation Categories

- Component
  - Convert QR to functionality
  - Apply Design pattern

- Architecture
  - Impose architectural pattern
  - Impose architectural style

Scope of impact

Transformation type

Added functionality, rules and/or constraints

Restructuring
Architectural Styles

- Are structures that recur and are used to solve specific types of problems. These include:
  - Pipes and Filters
  - Blackboard
  - Object-oriented

- System-level quality attributes can often be predicted based on the observation of certain architectural styles in a system’s architecture.

- In some cases, it is possible to moderate the degree to which a quality attribute is affected by using a variant of the style.

- It is also possible for a particular variant of a style to have both positive and negative affects on a given quality attribute.
## Architectural Styles

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Maintainability</th>
<th>Reliability</th>
<th>Safety</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipes and Filters</td>
<td>+ –</td>
<td>+ –</td>
<td>–</td>
<td>–</td>
<td>+</td>
</tr>
<tr>
<td>Blackboard</td>
<td>–</td>
<td>+</td>
<td>+ –</td>
<td>–</td>
<td>+ –</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>+ –</td>
<td>+ –</td>
<td>+ –</td>
<td>+</td>
<td>+ –</td>
</tr>
</tbody>
</table>
Architecture-Driven Component Development

- The goal for the embodiment phase of design is to either build or select components and connectors that possess the quality attributes identified during the architecting phase of development.

- Three types of components:
  - Custom built components
  - Reusable components
  - Commercial components
Custom Components

- Demands both time and money.

- Are most likely to pay off in cases of software that are:
  - Very unusual
  - Safety critical
  - Highly secure

- The component assembly will possess the quality attributes it was designed around.
Pre-Existing Components

- There are two main classes of pre-existing components:
  - Reusable components
  - Commercial components

- Is a fundamentally different problem than custom design.
  - The requirements to use specific components and component frameworks drive the architecture.
Reusable Components

- Can exist on a wide scale of reusability within any organization.

- They must be adapted
  - In most cases it will be necessary to create adaptors, often referred to as glue code.

- Are developed with reuse in mind.

- Product line development exemplifies the use of pre-planned reusable components.
Commercial Components

- Introduce a large degree of uncertainty.

- Tend to be
  - Complex
  - Idiosyncratic
  - Unstable
Component-Driven Architecture Development

- Constraints due to the use of pre-existing components:
  - *Design freedom is limited to component selection.*
  - Sufficient information about how a component will behave is not generally provided.
  - Component properties must be verified.
  - The framework into which components are to be plugged influences the architecture and the process by which the system is designed.
  - *Such components can not be optimized.*

- It is expected that *more reliable systems will be produced*, with greater speed and at lower expense due to the restrictions on design freedom.
Summary

- Components and Software Architectures form two sides of the same coin.

- Software architecture has multiple roles:
  - May be used for stakeholder-, expert-, or quality attribute-oriented assessment.
  - May be used for configuration management.
  - May be used to dynamically reorganize the system at run time (i.e. dynamic software architectures).

- Design of software architectures consists of three main phases:
  - Functionality-based architectural design
  - Software architecture assessment
  - Architecture transformation
CBSE and ADL

Before CBSE
- OO Technology: Weak global view

ADL: Global view
- An architecture description language is used to specify the structure of a system separately from its algorithmic aspects.
- Also known as Module Interconnection Language, Configuration Language

What is differences in CBSE
- 3rd party component vendor
  - Need for reusability
- How can we composite it?
Architecture & Definition/Use

- Module Interconnection Language & Software Architecture
Architecture & OO Design

- OO Design: Relationship between Classes
  - Method invocation
  - Protocols melted in methods.

- CBSE
  - For reusability, let’s focus on connection
Connection?

!!Wow So complex interaction!!
What is connection?
How about protocols?
Protocol?

We want to verify it!
We need formal basis.
OK! How about *process algebra*?
Architecture Description Language

✓ Examples:

➢ Conic Configuration Language
  (http://www-vs.informatik.uni-ulm.de/DOSinWWW/TextFiles/DPEnvironment/Conic.html)

➢ Darwin (http://www-dse.doc.ic.ac.uk/Research/Darwin/)

➢ Wright (http://www-2.cs.cmu.edu/~able/wright/)

➢ Polylith
  (http://www.cs.umd.edu/TRs/authors/James_M_Purtilo-no-abs.html)

➢ ...

✓ What is common thing?
General ADL

- Focus on interaction: connector
- Focus on component: component
- Focus on configuration: instances
- Analysis
WRIGHT :
“A Formal Basis for Architectural Connection”

- CMU : Robert Allen & David Garlan
- General Property
  - Semantic basis : a subset of CSP (*Communicating Sequential Processes*)
  - ADL
    - Component
    - Connection
    - Configuration
  - Analysis Coverage : Compatibility, Deadlock
  - Analysis Tool : Wright Toolset & FDR
WRIGHT : Overview of ADL

- Component & Connection Type
  - Component
    - Port & Spec
  - Connection
    - Role & Glue

- Instance

- Attachment
  - Role & Glue

System SimpleExample
  component Server =
    port provide [provide protocol]
    spec [Server specification]
  component Client =
    port request [request protocol]
    spec [Client specification]
  connector C-S-connector =
    role client [client protocol]
    role server [server protocol]
    glue [glue protocol]

Instances
  s: Server
  c: Client
  cs: C-S-connector

Attachments
  s.provide as cs.server;
  c.request as cs.client
end SimpleExample.
WRIGHT : Connector

- **Process Notation** : a subset of CSP
  - Processes and Events
  - Prefixing : $e \rightarrow P$
  - Alternative : “external choice” : $P \parallel Q$
  - Decision : “internal choice” : $P \parallel Q$
  - Named Processes : let ... in ...
  - Parallel Composition : $\parallel$ operator
WRIGHT : Connector

connector C-S-connector =
  role Client = (request!x → result?y → Client) ∩ §
  role Server = (invoke?x → return!y → Server) ∪ §
  glue = (Client.request?x → Server.invoke!x → Server.return?y → Client.result!y → glue)
    ∩ §

- Glue : Interaction between Components
  - Protocol

- Role : One peer obligation
connector Shared Data₁ =
  role User₁ = set → User₁ ⊔ get → User₁ ⊔ §
  role User₂ = set → User₂ ⊔ get → User₂ ⊔ §
  glue = User₁.set → glue ⊔ User₂.set → glue
       ⊔ User₁.get → glue ⊔ User₂.get → glue ⊔ §

connector Shared Data₂ =
  role Initializer =
    let A = set → A ⊔ get → A ⊔ §
    in set → A
  role User = set → User ⊔ get → User ⊔ §
  glue = let Continue = Initializer.set → Continue
       ⊔ User.set → Continue
       ⊔ Initializer.get → Continue
       ⊔ User.get → Continue ⊔ §
    in Initializer.set → Continue ⊔ §

connector Shared Data₃ =
  role Initializer =
    let A = set → A ⊔ get → A ⊔ §
    in set → A
  role User = set → User ⊔ get → User ⊔ §
  glue = let Continue = Initializer.set → Continue
       ⊔ User.set → Continue
       ⊔ Initializer.get → Continue
       ⊔ User.get → Continue ⊔ §
    in Initializer.set → Continue
       ⊔ User.set → Continue ⊔ §

connector Bogus =
  role User₁ = set → User₁ ⊔ get → User₁ ⊔ §
  role User₂ = set → User₂ ⊔ get → User₂ ⊔ §
  glue = let Continue = User₁.set → Continue
       ⊔ User₂.set → Continue
       ⊔ User₁.get → Continue
       ⊔ User₂.get → Continue ⊔ §
    in User₁.set → Continue
       ⊔ User₂.set → Continue ⊔ §
WRIGHT : Connector

connector Pipe =
  role Writer = write → Writer □ close → §
  role Reader = let ExitOnly = close → §
    in let DoRead = (read → Reader □ read-eof → ExitOnly)
    in DoRead □ ExitOnly
  glue = let ReadOnly = Reader.read → ReadOnly
    Reader.read-eof → Reader.close → §
    Reader.close → §
    in let WriteOnly = Writer.write → WriteOnly □ Writer.close → §
    in Writer.write → glue □ Reader.read → glue
      Writer.close → ReadOnly □ Reader.close → WriteOnly
The meaning of a connector

- Glue || (R1:R1||R2:R2||...||RN:RN)
- R1..N : Role

- $a_{Glue} = R1: \sum R2: \sum ... \cup RN: \sum \{V\}$
WRIGHT : Component - Port

$$\text{component DataUser =}
\text{port DataRead = get\rightarrow DataRead} \cap \notin \text{other ports...}$$

- Port : A component's role
  - Glue || (R1:P1 || R2:P2 ||…|| Rn:Pn)

- Port & Role : Compatible?
  - Not equality
  - More relaxed matching
  - Automatic Checking
WRIGHT : Analysis - Compatibility

- In CSP : refinement relation

- Process in CSP : (A, F, D)
  - A : alphabet of process
  - F : Failures
  - D : Divergences

- Refinement : $P \subseteq Q$
  - $A = A', \ F \subseteq F', \ D \subseteq D'$
  - $P = (e->P \ [\ ] \ f->p) \text{ and } Q = (e->Q) \text{ then } P \subseteq Q$
  - $P = (e->P \ [\ ] \ f->p) \text{ and } Q = (e->Q) \text{ then } P \not\subseteq Q$

- Make Deterministic version
WRIGHT : Analysis - Deadlock

- **What is deadlock.**
  - Not success finish case

- **Deadlock free semantic**
  - \((t, \text{ref}) \in \text{failures}(C) \text{ s.t. } \text{ref} = aC, \text{last}(t) = V\)

- **With compatibility**
  - Deadlock free connector C
  - Compatible connector C’
  - C’ is deadlock free

- **Local deadlock free \rightarrow Global deadlock free**
WRIGHT : Automatic Compatibility Checking

- Framework
  - Wright Spec
  - Wright Tool
  - FDR Notation
  - FDR Check

WRIGHT Specification

WRIGHT Tool

CSP

FDR
WRIGHT : Extending the Glue

- Trace specification

```plaintext
connector Pipe =
  role Writer = write!x⇒Reader ⊔ close⇒ §
  role Reader = let ExitOnly = close⇒ §
    in let DoRead = (read?x⇒Reader ⊔ read eof⇒ExitOnly)
    in DoRead ⊔ ExitOnly
  glue = let ReadOnly = Reader.ready⇒ReadOnly
    in read eof⇒Reader.close⇒ §
    in Reader.close⇒ §
  in let WriteOnly = Writer.write?x⇒WriteOnly ⊔ Writer.close⇒ §
  in Writer.write?x⇒glue ⊔ Reader.ready⇒glue
    ⊔ Writer.close⇒ReadOnly ⊔ Reader.close⇒WriteOnly
spec (Reader.read eof ⇒ (Writer.close ∧ #Reader.read = #Writer.write)) ∧
∀Reader.ready ⊔ (∃ Writer.write ?x ⊔ i = j ∧ x = y)
```
WRIGHT:
Why CSP/ Why not CSP?

- **Why CSP?**
  - Ability to capture certain critical properties
  - Simple but powerful form of composition
  - Automatic Analysis tool

- **Why not CSP?**
  - Architectural abstractions
  - Relationship between elements
Separate of concern

- Connector & Components
  - Maximize reusability
  - Intuitive Approach
WRIGHT : Conclusion

- The treatment of connectors as types
- Partitioning of connector descriptions into roles and glue
- The separation of the semantic definition into two parts: Protocols, auxiliary specification
- The application of formal machinery: automatic checking
ECOOP – WCOP 2004