Choosing Connectors

Software Architecture
Lecture 8
Role and Challenge of Software Connectors

How do we enable components A and B to interact?

- Attach adapter to A
- Introduce intermediate form
- Publish abstraction of A’s form
- Introduce intermediate form
- Transform on the fly
- Provide B with import/export converter
- Make B multilingual
- Maintain multiple versions of A
- Change A’s form to B’s common form
- Negotiate to find a common form for A and B
- Negotiate B’s “essence” from its packaging

What is the right answer?
How Does One Select a Connector?

- Determine a system’s interconnection and interaction needs
  - Software interconnection models can help
- Determine roles to be fulfilled by the system’s connectors
  - Communication, coordination, conversion, facilitation
- For each connector
  - Determine its appropriate type(s)
  - Determine its dimensions of interest
  - Select appropriate values for each dimension
- For multi-type, i.e., composite connectors
  - Determine the atomic connector compatibilities
Simple Example

- System components will execute in two processes on the same host
  - Mostly intra-process
  - Occasionally inter-process
- The interaction among the components is synchronous
- The components are primarily computation-intensive
  - There are some data storage needs, but those are secondary
Simple Example (cont’d)

- Select procedure call connectors for intra-process interaction
- Combine procedure call connectors with distributor connectors for inter-process interaction
  - RPC
- Select the values for the different connector dimensions
  - What are the appropriate values?
  - What values are imposed by your favorite programming language(s)?
### Procedure Call Connectors Revisited

<table>
<thead>
<tr>
<th>Service</th>
<th>Type</th>
<th>Dimension</th>
<th>Subdimension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Coordination</td>
<td>Procedure call</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameters</td>
<td>Data transfer</td>
<td>Reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Default values</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Keyword parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Inline parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invocation</td>
<td>Push from L to R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Push from R to L</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hash table</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entry point</td>
<td>Multiple</td>
<td>Method call</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single</td>
<td>Macro call</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inline</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>System call</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invocation</td>
<td>Explicit</td>
<td>Exceptions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Callbacks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Delegation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synchronicity</td>
<td>Implicit</td>
<td>Asynchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cardinality</td>
<td>Fan out</td>
<td>Private</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Protected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Public</td>
</tr>
</tbody>
</table>
## Distributor Connectors Revisited

<table>
<thead>
<tr>
<th>Service</th>
<th>Type</th>
<th>Dimension</th>
<th>Subdimension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facilitation</td>
<td>Distributor</td>
<td>Naming</td>
<td>Structure based</td>
<td>Hierarchical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Attribute based</td>
<td>Flat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery</td>
<td>Semantics</td>
<td>Best effort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Exactly once</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>At most once</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mechanism</td>
<td>At least once</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unicast</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Membership</td>
<td>Multicast</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Broadcast</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Path</td>
<td>Bounded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ad-hoc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Cached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dynamic</td>
</tr>
</tbody>
</table>
## Two Connector Types in Tandem

Select the appropriate values for PC and RPC!

### Table: Connector Types

<table>
<thead>
<tr>
<th>Service</th>
<th>Type</th>
<th>Dimension</th>
<th>Subdimension</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reference</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Default values</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Keyword parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Inline parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Push from L to R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Push from R to L</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hash table</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Return values</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invocation record</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entry point</td>
<td>Multiple</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invocation</td>
<td>Explicit</td>
<td>Implicit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synchronicity</td>
<td>Fan out</td>
<td>Fan in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cardinality</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accessibility</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naming</td>
<td>Structure based</td>
<td>Attribute based</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Routing</td>
<td>Semantics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distributor</td>
<td>Mechanism</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Membership</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Path</td>
<td></td>
</tr>
</tbody>
</table>

**Communication Coordination**: Procedure call

**Facilitation**: Distributor

---

Software Interconnection Models

- Interconnection models (IM) as defined by Perry
  - Unit interconnection
  - Syntactic interconnection
  - Semantic interconnection
- All three are present in each system
- Are all equally appropriate at architectural level?
Unit Interconnection

- Defines relations between system’s units
  - Units are components (modules or files)
  - Basic unit relationship is dependency
    - Unit-IM = ({units}, {“depends on”})
- Examples
  - Determining context of compilation
    - e.g., C preprocessor
    - IM = ({files}, {“include”})
  - Determining recompilation strategies
    - e.g., Make facility
    - IM = ({compile_units}, {“depends on”, “has changed”})
  - System modeling
    - e.g., RCS, DVS, SVS, SCCS
    - IM = ({systems, files}, {“is composed of”})
Unit Interconnection Characteristics

- Coarse-grain interconnections
  - At level of entire components
- Interconnections are static
- Does not describe component interactions
  - Focus is exclusively on dependencies
Syntactic Interconnection

- Describes relations among syntactic elements of programming languages
  - Variable definition/use
  - Method definition/invocation
    - $IM = (\{\text{methods, types, variables, locations}\},$
    - $\{\text{is def at}'', \text{is set at}'', \text{is used at}'',$
    - $\text{is del from}'', \text{is changed to}'', \text{is added to}'')$

- Examples
  - Automated software change management
    - e.g., Interlisp’s masterscope
  - Static analysis
    - e.g., Detection of unreachable code by compilers
  - Smart recompilation
    - Changes inside unit $\rightarrow$ recompilation of only the changes
  - System modeling
    - Finer level of granularity than unit-IM
Syntactic Interconnection Characteristics

- Finer-grain interconnections
  - At level of individual syntactic objects
- Interconnections are static & dynamic
- Incomplete interconnection specification
  - Valid syntactic interconnections may not be allowed by semantics
  - Operation ordering, communication transactions
    - e.g., Pop on an empty stack
  - Violation of (intended) operation semantics
    - e.g., Trying to use calendar `add` operation to add integers
Semantic Interconnection

- Expresses how system components are meant to be used
  - Component designers’ intentions
- Captures how system components are actually used
  - Component users’ (i.e., system builders’) intention
- Interconnection semantics can be formally specified
  - Pre- & post-conditions
  - Dynamic interaction protocols (e.g. CSP, FSM)
    - $IM = \{\{\text{methods, types, variables, ...}, \text{predicates}\},$
      \{“is set at”, “is used at”, “calls”, “called by”, ...
      “satisfies”\}\)
Example of Semantic Interconnection

connector Pipe =
  role Writer = write \rightarrow Writer \sqcup close \rightarrow \checkmark
  role Reader =
    let ExitOnly = close \rightarrow \checkmark
    in let DoRead = (read \rightarrow Reader
    read-eof \rightarrow ExitOnly)
    in DoRead \sqcup ExitOnly
  glue = let ReadOnly = Reader.read \rightarrow Reader.close \rightarrow \checkmark
       Reader.close \rightarrow \checkmark
    in let WriteOnly = Writer.write \rightarrow Writer.close \rightarrow \checkmark
    in Writer.write \rightarrow glue
    Reader.read \rightarrow glue
    Writer.close \rightarrow ReadOnly
    Reader.close \rightarrow WriteOnly
Semantic Interconnection Characteristics

- Builds on syntactic interconnections
- Interconnections are static & dynamic
- Complete interconnection specification
  - Specifies both syntactic & semantic interconnection validity
- Necessary at level of architectures
  - Large components
  - Complex interactions
  - Heterogeneity
  - Component reuse
- What about ensuring other properties of interaction?
  - Robustness, reliability, security, availability, ...
Composing Basic Connectors

- In many systems a connector of multiple types may be required to service (a subset of) the components.
- All connectors cannot be composed:
  - Some are naturally interoperable
  - Some are incompatible
  - All are likely to require trade-offs
- The composition can be considered at the level of connector type dimensions and subdimensions.
Connector Dimension Inter-Relationships

- **Requires** – &
  - Choice of one dimension mandates the choice of another

- **Prohibits** – ❌
  - Two dimensions can never be composed into a single connector

- **Restricts** – ⊗
  - Dimensions are not always required to be used together
  - Certain dimension combinations may be invalid

- **Cautions** – ⚠️
  - Combinations may result in unstable or unreliable connectors
# Dimension Inter-Relationships in a Nutshell

![Dimension Inter-Relationships in a Nutshell](image.png)
Well Known Composite Connectors

- Grid connectors (e.g., Globus)
  - Procedure call
  - Data access
  - Stream
  - Distributor
- Peer-to-peer connectors (e.g., Bittorrent)
  - Arbitrator
  - Data access
  - Stream
  - Distributor
- Client-server connectors
- Event-based connectors