Software Interconnection Models

- Interconnection Models (IM) as defined by Perry
  - unit interconnection
  - syntactic interconnection
  - semantic interconnection

- All three are present in a system
  → Which is the most appropriate at the architectural level?

Unit Interconnection

- Defines relationships between a system’s units
  - units are components (modules or files)
  - basic unit relationship is dependency
    - \( UnitIM = (\{units\}, \{“depends on”\}) \)

- Examples
  - determining the context of compilation
    - e.g., the 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, CVS, SCCS
    - \( IM = (\{systems, files\}, \{“is composed of”\}) \)
Unit Interconnection Characteristics

- Coarse-grain interconnections
  - at the level of entire components
- Interconnections are static
  - applicable on only one of Kruchten’s 4+1 views
- Does not describe component interactions
  - focus is exclusively on dependencies
- Applicable on implemented modules only

---

Syntactic Interconnection

- Describes relations among syntactic elements of PLs
  - variable definition/use
  - method definition/invocation
    - \( IM = ( \{ methods, types, variables, locations \}, \{ \text{is def at}, \text{is set at}, \text{is used at}, \text{is del from}, \text{is changed to}, \text{is added to} \} ) \)
- Examples
  - automated software change management
    - e.g., Interlisp’s Masterscope
  - static analysis
    - e.g., detection of unreachable code by compilers
  - smart recompilation
    - changes inside methods localize recompilation
  - system modeling
    - finer level of granularity than the unit IM
Syntactic Interconnection Characteristics

- Finer-grain interconnections
  - at the level of individual syntactic objects
- Interconnections are static and dynamic
- Applicable on conceptual and implemented modules
- Incomplete interconnection specification
  - valid syntactic interconnections may not really be allowed
    - operation ordering, communication transactions
      - e.g., a pop on an empty stack
    - violation of (intended) operation semantics
      - e.g., using a 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 used
  - component users’ (i.e., system builders’) intentions
- Interconnection semantics can be formally specified
  - pre- and postconditions
  - dynamic interaction protocols (e.g., CSP, FSM)
    - $IM = ( \{ \text{methods, types, variables, }\ldots, \text{predicates}\),
      \{“is set at”, “is used at”, “calls”, “called by”, \ldots, “satisfies”\})$
Example of Semantic Interconnection

```hs
connector Pipe =
  role Writer = write \rightarrow Writer \cap close \rightarrow /
  role Reader =
    let ExitOnly = close \rightarrow /
    in let DoRead = (read \rightarrow Reader
data-read-eof \rightarrow ExitOnly)
    in DoRead \cap ExitOnly
  glue = let ReadOnly = Reader.read \rightarrow ReadOnly
         Reader.read-eof \rightarrow Reader.close \rightarrow /
         Reader.close \rightarrow /
    in let WriteOnly = Writer.write \rightarrow WriteOnly
         Writer.close \rightarrow /
    in Writer.write \rightarrow glue
         Reader.read \rightarrow glue
         Reader.close \rightarrow ReadOnly
         ReadWrite.close \rightarrow WriteOnly
```

Semantic Interconnection Characteristics

- Builds on syntactic interconnections
- Interconnections are static and dynamic
- Applicable on conceptual and implemented modules
- Complete interconnection specification
  - specifies both syntactic and semantic interconnection validity
- Necessary at the level of architectures
  - large components
  - complex interactions
  - heterogeneity
  - component reuse
Software Connectors

- A connector is an architectural element that models
  - interactions among components
  - rules that govern those interactions

- Simple interactions
  - procedure calls
  - shared variable access

- Complex and semantically rich interactions
  - client-server protocols
  - database access protocols
  - asynchronous event multicast

Multiple Connectors in a Single System
**Implemented vs. Conceptual Connectors**

- Connectors in software system implementations
  - no code
  - no identity
  - typically do not correspond to compilation units
  - distributed implementation
    - across multiple modules
    - across interaction mechanisms

- Connectors in software architectures
  - first-class entities
  - have identity
  - describe all system interaction
  \(\rightarrow\) entitled to their own specifications and abstractions

**Reasons for Treating Connectors Independently**

- Connector \(\neq\) Component
  - components provide application-specific functionality
  - connectors provide application-independent interaction mechanisms

- Interaction abstraction and/or parameterization

- Specification of complex interactions
  - binary vs. n-ary
  - asymmetric vs. symmetric
  - interaction protocols

- Localization of interaction definition

- Extra-component system (interaction) information

- Component independence

- Connector independence

- Component interaction flexibility
Benefits of First-Class Connectors

- Separate computation from communication
- Minimize component interdependencies
- Support software evolution
  - at component-, connector-, and system-level
- Potential for supporting dynamism
- Facilitate heterogeneity
- Become points of distribution
- Aid system analysis and testing

Software Connector Roles

- A connector is a locus of interaction among a set of components
- A connector has a protocol specification that defines its properties
  - types of interfaces it is able to mediate
  - assurances about interaction properties
  - rules about interaction ordering
  - interaction commitments (e.g., performance)

- Connector roles
  - communication
  - mediation
  - coordination
Connectors as Communicators

- The role typically associated with connectors
- Different communication mechanisms
  - procedure call, RPC, shared data access, message passing
  - constraints on communication structure/direction — pipes
  - constraints on quality of service — persistence
- Separate communication from computation
- May influence non-functional system characteristics
  - e.g., performance, scalability, security

Connectors as Mediators

- Govern access to shared information
- Determine allowed functionality and interactions
  - e.g., reader/writer policies
- Provide synchronization mechanisms
  - critical sections
  - monitors
- Separate mediation from computation
- Separate mediation from communication
- Allows composition of mediation and communication connectors
  → flexibility
Connectors as Coordinators

- Determine computation control
- Control delivery of data
- Control loci of execution
- Separates control from computation
- Orthogonal to communication and mediation
  - there are elements of control in communication and mediation

Additional Dimensions of Connector Variability

<table>
<thead>
<tr>
<th>Type of Interaction</th>
<th>Degree of Concurrency</th>
<th>Degree of Information Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Passing</td>
<td>Single Thread</td>
<td>Partial Communication</td>
</tr>
<tr>
<td>Pipe</td>
<td>Inter Thread</td>
<td>Full Communication</td>
</tr>
<tr>
<td>Procedure Call</td>
<td>Inter Process</td>
<td>No Interaction</td>
</tr>
<tr>
<td>No Interaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full Communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial Communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Thread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter Thread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter Process</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Connectors as Facilitators of Distribution and Heterogeneity

Connectors as Facilitators of Distribution and Heterogeneity (2)
Interaction Mismatches

- A direct by-product of reuse and heterogeneity
- Components must be undisturbed
  → The responsibility is connectors’
- Interaction mismatch handling techniques
  - pairwise information reformatting/conversion
  - interchange to and from a single shared format
  - data conversion modules
    - buffers
    - wrappers
    - adaptors
    - domain translators in C2

Discussion

- Connectors allow modeling of arbitrarily complex interactions
- Connector flexibility aids system evolution
  - component addition, removal, replacement, reconnection
- Support for connector interchange is desired
  - aids system evolution
  - may not affecting system functionality
- Libraries of OTS connector implementations allow developers to focus on application-specific issues
- Difficulties
  - rigid connectors
  - connector “dispersion” in implementations
- Key issue
  → effective connector protocol models vs. flexibility