the

Architect's Automated Assistant

Ahmed A. Abd-Allah

Domain-Driven Style Composition

Satellite

Satellite Ground Station

Users

Pipe and Filter

Event-Based

Design of a Satellite Ground Station
Recognition of Style Composition Mismatches

- Problems:
  - architectural styles focus on different properties
  - finding appropriate representation languages
  - composition is achievable in multiple ways

- Approach taken:
  - model a few "pure" architectural styles
  - model an architectural style space
    ==> basis for a uniform representation
  - model architectural composition operations
    ==> basis for identifying possible mismatches
  - implement a prototype based on the model
Towards a Uniform Representation

- Identify core set of base elements:

1. **Port**: typically associated with a control component, and represents the latter's entry and exit points for data components during data transfers.
2. **Data Component**: models data that is used to store state or to be passed around in data transfers (communication).
3. **Control Component**: models data that is executed by the underlying machine and which can initiate (and respond to) data and control transfers. It is assumed to have a single thread of control.
4. **Object**: an encapsulation of data components with control components.
5. **Data Connector**: models the potential for two or more control components to engage in data transfers amongst themselves.
6. **Control Connector**: models the potential for two or more control components to engage in control transfers (possibly with data) amongst themselves.
7. **Trigger**: the association of an action with the reception of a particular data component.
8. **System**: any collection of control components which can potentially engage in a control or data transfer - i.e. there is at least one control or data connector joining two components.

- Base elements are refined to produce style-specific elements (pipes, procedures, etc.)

Architectural Element Taxonomy

- Architect can refine these further for a particular system
Representing Style-Specific Systems

- Constrain the elements to capture style

**Example: Pipe & Filter System**

- Representation supports analysis during composition

Towards an Architectural Style Space

- Can describe *architectural space* of conceptual features.

<table>
<thead>
<tr>
<th>Element</th>
<th>Conceptual Feature</th>
<th>Choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>concurrency of computations</td>
<td>single, multiple</td>
</tr>
<tr>
<td>System</td>
<td>dynamism of computations</td>
<td>static, dynamic</td>
</tr>
<tr>
<td>System</td>
<td>(supported control transfers)</td>
<td>(calls, spawns)</td>
</tr>
<tr>
<td>Data</td>
<td>supported data transfers</td>
<td>implicit global data distributor, explicit</td>
</tr>
<tr>
<td>Connector</td>
<td></td>
<td>data channels, shared variables</td>
</tr>
<tr>
<td>System</td>
<td>locality of components/connectors</td>
<td>local, distributed</td>
</tr>
<tr>
<td>System</td>
<td>component groupings</td>
<td>layered, monolithic</td>
</tr>
<tr>
<td>Trigger</td>
<td>triggered actions</td>
<td>yes/no</td>
</tr>
<tr>
<td>Object</td>
<td>encapsulation</td>
<td>yes/no</td>
</tr>
</tbody>
</table>
Instantiating a few Styles in the Space

- Pipes & Filters, Main/Subroutine, Distributed Processes, Event-Based

<table>
<thead>
<tr>
<th>Style Feature</th>
<th>P/F</th>
<th>M/S</th>
<th>D/P</th>
<th>E/B</th>
</tr>
</thead>
<tbody>
<tr>
<td>concurrency of computations</td>
<td>multiple</td>
<td>single</td>
<td>multiple</td>
<td>multiple</td>
</tr>
<tr>
<td>dynamism of computations</td>
<td>static</td>
<td>static</td>
<td>dynamic</td>
<td>dynamic</td>
</tr>
<tr>
<td>supported control transfers</td>
<td>(none)</td>
<td>(call)</td>
<td>(call, spawn)</td>
<td>(call, spawn)</td>
</tr>
<tr>
<td>supported data transfers</td>
<td>exp. data chan.</td>
<td>shared var.</td>
<td>exp. data chan.</td>
<td>imp. global data dist., shared var.</td>
</tr>
<tr>
<td>locality of components/connectors</td>
<td>local</td>
<td>local</td>
<td>distributed</td>
<td>local</td>
</tr>
<tr>
<td>component groupings</td>
<td>monolithic</td>
<td>monolithic</td>
<td>monolithic</td>
<td>monolithic</td>
</tr>
<tr>
<td>triggered data channels</td>
<td>N/A</td>
<td>N/A</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>encapsulation</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Multiplicity of Architectural Compositions

- Several operations used to “compose” architecturally:

1. Connecting two or more systems via data and control transfers
   - calls, spawns, data channels, shared variables

2. Wrapping a system into a single component, then calling or spawning that component in another system

3. Adding an interface to a system in one style so that it appears as a system in another style

```
Call     Spawn     Data Channel
        Shared Data
StyleA System

StyleB System

InterfaceAB
```

(1) (2) (3)
**Tool Overview**

- The Architect's Automated Assistant (AAA) supports
  - specification of software system architectures
  - specification of architectural composition
  - analysis for architectural constraint mismatches
- Systems are specified in terms of architectural styles
  - main/subroutine, pipe & filter, distributed processes, event-based, object-oriented, blackboard, ...
- COTS packages can be expressed as very specific styles
  - TRW's UNAS, HP's Softbench

**Scenario Background**

- Based on Garlan et. al. "Architectural Mismatch" experience (IEEE Software, 11/95)
- Project: build a CASE tool (Aesop) using four COTS packages
  1. OBST, object-oriented database (Forschungszentrum Informatik)
  2. InterViews, event-based GUI builder (Stanford)
  3. Softbench, event-based tool integrator (HP)
  4. MIG, Mach RPC Interface generator (CMU)
**Project Results**

- Much harder than anticipated
  - Anticipated: Six months, one person-year
  - Actual: Two years, five person-years
- General problems
  - excessive code size
  - poor performance
  - need to modify COTS packages
  - need to reinvent existing function
  - unnecessarily complicated tools
  - error-prone construction process

**Technical Details of AAA**

- Internal knowledge base modeled in Z
- Tool implemented in Prolog
- Runs on Sun workstations, SunOS 4.1.3
Catching Some of the Problems Early in the Lifecycle

- AAA can be used to specify the architecture of Aesop
- Four COTS packages are specializations of architectural styles
  - InterViews, Softbench, MIG: event-based
  - OBST: object-oriented database
- Tool analyzes composition and reports potential architectural mismatches

Potential Mismatches (I)

- Softbench, InterViews, MIG all event-based, but with incompatible event sets. Modify InterViews to be compatible with Softbench; no time for MIG.
  => need to modify COTS packages
- OBST large library of standard object classes must be installed even if not needed.
  => excessive code size
- Softbench requires all components to link X library.
  => excessive code size
- MIG event data based on C structs/arrays, and Softbench event data based on ASCII string. Required a realtime translator.
  => poor performance
Potential Mismatches (II)

- OBST database assumes no concurrent accesses. Required building transaction manager.
  
  ==> need to modify COTS package

- Softbench does not provide blocking on 'request' & 'reply' event pairs. Mach RPC (MIG) introduced to satisfy this need for procedure call semantics.
  
  ==> unnecessarily complicated tools, excessive code size

- InterViews does not allow independent manipulation of window children. Duplicate datastructures coded.
  
  ==> need to reinvent existing function

- System dependencies for building Aesop out of four COTS packages were tangled. Required manually aiding an automated build process.
  
  ==> error-prone construction process