Software Architecture
Dependence Analysis

Ground Systems Architecture Workshop

Debra J. Richardson
Judith A. Stafford
Alexander L. Wolf

University of California at Irvine
University of Colorado at Boulder
Overall Problem

- Increasing size and complexity of software
  - Intractable
  - Increased expense of design faults

- Some Related Issues
  - Legacy Code
    » What do we want to do with it?
  - System evolution
    » How can we best maintain and enhance existing systems?
  - Future directions
    » How can we minimize these problems in the future?
Overall Solution

- **Automated dependence analysis of software architectures**
  - based on ADL descriptions of software architecture
    - basic architectural elements
    - support modeling of behavior and structure
  - to enable early, high-level analysis accessible to developers.

- **First question**: What are the *meaning* and *application* of dependence analysis for software architectures?
Architecture Dependence Analysis

Architectural Relationships
What types of relationships exist at the architecture level?

Architecture-based Questions
What types of dependence-related questions are interesting at the architecture level?

Architecture Dependence Analysis
How do we exploit these relationships to answer these questions?
The system is comprised of a client and a server. The client directs the server to do something.

**Box and Arrow Diagrams**

Client  A → B  Server

**Natural Language Text**

The system is comprised of a client and a server. The client directs the server to do something.

**ADL Specifications**

```
component Client
  { Out: A;
    Behavior
    Send A;
  }
component Server
  { In: B;
    Behavior
    When B then
    DOSOMETHING;
  }
architecture Client-Server
  { Server.A to Client.B; }
```
Program Dependence Analysis

- Centered on Statements and Variables
- Sources of Dependence Relationships
  - Control and data flow
- Example Uses
  - Code optimization
  - Program understanding
  - Testing and analysis
  - Debugging
  - Impact analysis, change management, maintenance
Architecture Dependence Analysis

- **Centered on Components and Interactions**
- **Sources of Dependence Relationships**
  - Structure (include, import/export, inheritance)
  - Behavior (input/output, temporal, causal)
  - Non-functional (safety level, performance requirements)
- **Example Uses**
  - Impact analysis
  - Architecture-based Integration testing
  - Architecture/Program debugging
  - Workspace management
  - Dynamic Addition/Deletion of Components
  - Reuse
  - Safety
  - Security
  - Regression Testing
  - ...
Some Architectural Relationships

Temporal: The car must be in park before the ignition is allowed to function.

Causal: If a plane is within range set system on alert.

Safety level: No Component may be impacted by a less safety critical component.

Combined: A includes B which inherits from C
Tabular Representation of Dependence Relationship

Table frame is built by recording the ports

ADL Specifications

component Client
{ Out: A;
  Behavior
  Send A; }
component Server
{ In: B;
  Behavior
  When B then
  DOSOMETHING; }
architecture Client-Server
{ server: Server;
  client: Client;
  connect
  server.A to client.B; }

Relationships are recorded in the cells
What caused this to happen?
Chaining

- **Chain-links** represent direct dependencies between components.
- **Chains** represent indirect relationships among components.
- Chaining is
  - the construction of chains.
  - a means for performing software architecture dependence analysis.
  - a way to answer questions about software architectures.
Inspiration for Chaining

◆ **ADAGE Avionics System**
  - Complex architecture described in Rapide
    » three-level nested architecture
    » 30 components
    » 100 ports
  - Informal chaining used to isolate cause of system’s incorrect emission of a warning
  - Reduced mental debugging to 25% of system’s components
Chains - A Component-Centric View

If this component is replaced, what components will need to be retested?

What components could have contributed to a failure in this component?

If I am going to work on this component, what other components do I want to have immediately available?
Planned Activities

- Investigate Coupling of Large Systems
  - How effective can we expect chaining to be?
- In-Depth Comparison of Expressiveness and Modeling Capabilities of ADLs
  - Which relationships are modeled in which ADLs?
- Refinement of Chaining Technique
  - How can we make chains more precise?
- Development of Aladdin Chaining Tool
- Experiment with Chaining
Experience with Chaining

- **Gas Station Example**
  - Simple architecture described in Rapide
  - 1 operator, 1 pump, and 2 customers
type Dollars is integer; - enum 0, 1, 2, 3 end enum;
type Gallons is integer; - enum 0, 1, 2, 3 end enum;

type Customer is interface
  action in	Okay(), Change(Cost : Dollars);
  out 	Pre-Pay(Cost : Dollars), Okay(), Turn_On(), Walk(), Turn_Off();
behavior
  D : Dollars is 10;
begin
  start ||> Pre_Pay(D);;
  Okay ||> Walk;;
  Walk ||> Turn_On;;
end Customer;

type Operator is interface

type Pump is interface

architecture gas_station() return root is
  O : Operator;
  P : Pump;
  C1, C2 : Customer;
connect
  (?C : Customer; ?X : Dollars) ?C.Pre_Pay(?X) ||> O.Request(?X);
  (?X : Dollars) O.Schedule(?X) ||> P.Activate(?X);
  (?X : Dollars) O.Schedule(?X) ||> C1.Okay;
  (?C : Customer) ?C.Turn_On ||> P.On;
  (?C : Customer) ?C.Turn_Off ||> P.Off;
  (?X : Gallons; ?Y : Dollars)P.Report(?X, ?Y) ||> O.Result(?Y);
end gas_station;
Gas Station Matrix

<table>
<thead>
<tr>
<th>Operator</th>
<th>Pump</th>
<th>Customer1</th>
<th>Customer2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>Sch</td>
<td>Rem</td>
<td>Req</td>
<td>Res</td>
</tr>
<tr>
<td>Remitt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>On</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activate</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

↓↓ Rapide *agent* connection: Models new thread of control for each triggering.

=> Rapide *pipe* connection: Models single thread of control thus creating additional dependencies on prior triggerings of the rule.
Experience with Chaining

- **Gas Station Example**
  - Table representation used to
    » detect anomaly
  - Chaining used to
    » determine components that will be affected if pump report is altered
    » identify fault that allows only first customer to pump gas
Gas Station - Anomaly

<table>
<thead>
<tr>
<th>Operator</th>
<th>Pump</th>
<th>Customer1</th>
<th>Customer2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Out</td>
<td>Out</td>
<td>Out</td>
</tr>
<tr>
<td>Sch</td>
<td>Rem</td>
<td>Req</td>
<td>Res</td>
</tr>
<tr>
<td>In</td>
<td>In</td>
<td>Rep</td>
<td>On</td>
</tr>
<tr>
<td>Out</td>
<td>Out</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>PP</td>
<td>T_On</td>
<td>Walk</td>
<td>T_Off</td>
</tr>
<tr>
<td>Okay</td>
<td>Chg</td>
<td>start</td>
<td></td>
</tr>
<tr>
<td>OK</td>
<td>Chg</td>
<td>start</td>
<td></td>
</tr>
</tbody>
</table>

---

**Schedule**

- **Remit**
- **Request**
- **Result**
- **Report**
- **On**
- **Off**
- **Activate**

**Operator**

- **Out**
- **In**

**Pump**

- **Pre_Pay**
- **Turn_On**
- **Walk**
- **Turn_Off**
- **Okay**
- **Change**
- **Start**

**Customer1**

- **Out**
- **In**

**Customer2**

- **Out**
- **In**
# Gas Station - Impact Analysis

<table>
<thead>
<tr>
<th>Operator</th>
<th>Pump</th>
<th>Customer1</th>
<th>Customer2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>Sch</td>
<td>Rem</td>
<td>Req</td>
<td>Res</td>
</tr>
<tr>
<td>Remit</td>
<td>Request</td>
<td>Result</td>
<td>Report</td>
</tr>
<tr>
<td>On</td>
<td>=&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off</td>
<td>=&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activate</td>
<td>=&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre_Pay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turn_On</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turn_Off</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Okay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre_Pay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turn_On</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turn_Off</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Okay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GSAW 98
Gas Station - Architecture Debugging