Extending Win Win to Address Architecture and Rationale Capture

Ellis Horowitz, Ming June Lee, USC-CSE
Frank Belz, TRW
USC-CSE Annual Research Review
March 11, 1997
Outline

- Introduction -- Barry Boehm
- Win Win Extensions: Usage Scenario -- Ming June Lee
- Collaboration and Rationale Capture Lessons Learned -- Frank Belz
- Win Win Extensions: Software Development -- Ellis Horowitz
Context

- DARPA EDCS (Evolutionary Design of Complex Software) program sponsoring Win Win research extensions
  - Integration with architecture-based tradeoff analysis tools for negotiation support
  - Capture of decision rationale for future use
  - Usage experimentation in satellite ground station (SGS) context
  - In concert with Aerospace, TRW
- Also collaborating on other DARPA-EDCS projects with Northrop Grumman, Lockheed Martin, SEI
Initial Prototypes and Experiments

- Develop prototype architecture capture tool to support negotiation
- Integrate with Win Win via artifact attachments
- Experiment with distributed negotiation of SGS architecture-based decision issues
  - Asynchronously at USC, Aerospace, TRW
  - Capture decision rationale
- Develop use cases for captured rationale
  - Determine usage lessons learned
Overview of Software Development in the CSE

WinWin System for supporting the capture of system requirements
Available on Sparc + Java

COCOMO.II
for estimating software effort and schedule
Sparc + Windows + Java

ACT I
for capturing and analyzing software architectures
Planned on Sparc

Other tools
Expressing Dynamic Properties of Pipe-and-Filter

for all data sent out a filter to a port and along a pipe, the same data arrives at the adjacent port and then is sent to the attached Filter as input

for (k = first("Filter"), k != 0, k = next("Filter") )
{
    port1 = portlist(k,2); pipe = linelist(port1,1);
    port2 = portlist(pipe,2); filter2 = boxlist(port2,1);
    if ! ( k.OutputData == port1.Data &&
         port1.Data == pipe.Data &&
         pipe.Data == port2.Data &&
         port2.Data == filter2.InputData)
        msg = "error - Pipe-Filter data transfer is not correct";
}
Expressing Structural Properties

/* All Filters have two Ports and no Pipes */
for (k = first("Filter"), k != 0, k = next("Filter") )
    { if (length(LineList(k)) != 0 || length(PortList(k)) != 2)
        msg="error - filter is misconstructed"; }

/* All Pipes attach to two Ports, but to no Filters */
for (k = first("Pipe"), k != 0, k = next("Pipe") )
    { if !(length(BoxList(k)) == 0 && length(PortList(k)) == 2)
        msg="error - pipe is misconstructed"; }

/* All Ports are attached to one Filter and one Line */
for (k = first("Port"), k != 0, k = next("Port") )
    { if !(length(BoxList(k)) == 1 && length(LineList(k)) == 1)
        msg="error - port is misconstructed"; }
ACT I Description of Pipe_and-Filter

Table 1: Attributes of Pipe-and-Filter

<table>
<thead>
<tr>
<th>Box 1</th>
<th>Box 2</th>
<th>Port 3</th>
<th>Port 4</th>
<th>Line5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Filter</td>
<td>Type: Filter</td>
<td>Type: port</td>
<td>Type: port</td>
<td>Type: Pipe</td>
</tr>
<tr>
<td>ID: 1</td>
<td>ID: 2</td>
<td>ID: 3</td>
<td>ID: 4</td>
<td>ID: 5</td>
</tr>
<tr>
<td>LineList: Empty</td>
<td>LineList: Empty</td>
<td>BoxList: &quot;1&quot;</td>
<td>BoxList: &quot;2&quot;</td>
<td>BoxList: Empty</td>
</tr>
<tr>
<td>PortList:&quot;3&quot;</td>
<td>PortList:&quot;4&quot;</td>
<td>LineList:&quot;5&quot;</td>
<td>LineList:&quot;5&quot;</td>
<td>PortList:&quot;3,4&quot;</td>
</tr>
<tr>
<td>InputData</td>
<td>InputData</td>
<td>Direction: &quot;out&quot;</td>
<td>Direction: &quot;in&quot;</td>
<td>Direction: &quot;forward&quot;</td>
</tr>
<tr>
<td>OutputData</td>
<td>OutputData</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
</tbody>
</table>
Killed = Source.write -> Killed [] Source.close -> !

Constraints
  forall c: Connectors | Type(c) = Pipe  /*All connectors have type Pipe*/
  forall c: Components | Filter(c)  /*All ports are either for input or for output*/
      where Filter(c:Component) = forall p : Ports(c) | Type(p) = DataInput
         or Type(p) = DataOutput

End Style
Pipe-and-Filter Style due to D. Garlan (CMU) et al in Wright (IEEE Software 2/97)

Style pipe-and-filter

Interface Type DataInput = (read -> (data?x -> DataInput
               [] end-of-data -> close -> !))

Interface Type DataOutput = write -> DataOutput || close -> !

Connector Pipe

Role Source = DataOutput
Role Sink = DataInput
Glue = Buf<>
where

Buf<> = Source.write?x -> Buf<x> [] Source.close -> Close<>
BufS<x> = Source.write?y -> Buf<y>S<x>
               []Source.close -> closedS<x>
               []Sink.read ->Sink.data!x->BufS
               []Sink.close -> Killed
ClosedS<X> = Sink.read -> Sink.data!x -> ClosedS
               [] Sink.Close -> !
Closed<> = Sink.read -> Sink.end-of-data -> Sink.close ->!
A filter is a processor that has an input socket (e.g. stdin) and an output socket (e.g. stdout). Data enters the input socket, is processed, and the result is sent to the output socket. The data travels along the pipe to the next filter.
Focus of *ACT I*

<table>
<thead>
<tr>
<th>System</th>
<th>Authors</th>
<th>Graphical Input</th>
<th>Underlying Model</th>
<th>Code Simulation</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ACT I</em></td>
<td>Horowitz et al</td>
<td>YES</td>
<td>Java</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

In addition, *ACT I* is different from other tools in that it:

- has explicit support for defining and maintaining architectural styles;
- supports the development of a system by the composition of styles;
- uses Java for describing relationships among styles and style components;
There already are other architecture design tools:

<table>
<thead>
<tr>
<th>System</th>
<th>Authors</th>
<th>Graphical Input</th>
<th>Underlying Model</th>
<th>Simulation</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wright</td>
<td>Garlan et al</td>
<td>No</td>
<td>CSP</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Unicon</td>
<td>Shaw et al</td>
<td>Yes</td>
<td>PropertyList</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Rapide</td>
<td>Luckham et al</td>
<td>No</td>
<td>Posets</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Aesop</td>
<td>Garlan et al</td>
<td>Yes</td>
<td>PropertyList</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>*Statemate</td>
<td>Hare et al</td>
<td>Yes</td>
<td>StateCharts</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*ObjecTime</td>
<td>Selic et al</td>
<td>Yes</td>
<td>ROOM</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* commercial products
Architecture Capture Tool Version I - ACT I

- The Center for Software Engineering's latest software effort

*Our purpose:* to investigate the value of computer software architectures as a way to assist the software development process;

It has been proposed that software architectures can be used to assist:

- in the design of a software system
- in the reuse of software designs
- in *simulation* of designs
- in *verification* of designs
- in *documentation* of designs
Rejected Strategy: Re-write the COCOMO legacy code entirely in Java

Pro
- one version for all platforms
- can be used as an applet or a stand-alone program

Con
- Debugged legacy code is lost
- Java is slow, legacy code is efficient
- No obvious mapping between C and Java
- Java programs need a run-time system
Java COCOMO.II Main Screen

The user interface across platforms (Sparc, Windows95, Java) is preserved.
COCOMO.II is our Software Cost Estimation Tool

Modules
Source Code Estimates
produces
Effort + Schedule
LAN + JAVA Architecture of WINWIN

- **Six stakeholders**
  - local WinWin data
  - database interface
  - Motif user interface
  - database server interface
  - database server
  - WinWin
  - Java/WinWin user interface
  - Internet
  - virtual winwin
    - DB interface
    - DB server interface
    - data structures
  - WinWin

Diagram showing the architecture of the WinWin program with interfaces and data flows.
LAN Architecture of WINWIN

![Diagram of WINWIN architecture]

- **Local WinWin data**
- **WinWin Program**
- **Motif user interface**
- **Database interface**
- **Database server**
- **WinWin**
- **Data structures**

**Six stakeholders**
Remote User Interface Architecture

devised with June Sup Lee
Center for Software Engineering

WWWWeb Architecture Strategies

Client-side Architectural Style

Server Machine
- Web docs
- Web Server

Internet/Intranet
- request
- download Java Program data

Client Machine
- Browser Program
- Execute Program

Server-side Architectural Style

Server Machine
- Execute Program
- CGI Script
- Web Server

Internet/Intranet
- request
- response
- display output

Client Machine
- Browser Program
Advantages of Porting our Software to the Web

- Platform Independence - our affiliates use many different machines

- Wide area access - collaboration was restricted to a LAN

- Ability to handle hypertext and hypermedia - negotiation typically involves spreadsheets, graphs, and even video and sound
Strategies for Transforming our Software to the Web

WinWin: Port to Java

COCOMO.II: Port to Java

Specification of the Architecture Capture Tool (ACT I)
Software Development
at the
Center for Software Engineering

Prof. Ellis Horowitz
Presentation at CSE Research Review
March 11, 1997