Coupling Architectural Discovery and Recovery to Stem Architectural Erosion

Nenad Medvidovic
Center for Software Engineering
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781 U.S.A.
neno@usc.edu
Why Architectures Erode

It’s easier than the alternative!
What Erosion Means to Developers

- Architectural models cannot be trusted

- But... requirements and code still might
Software Requirements

- High level
- Possibly imprecise, incomplete, ambiguous
- Meaningful to a wide range of human stakeholders
  - Often not amenable to manipulation by software tools
- Rich in insight and rationale
- “Cloak” architectural information
  - Components, interaction mechanisms, topology, properties
  - Suggest suitable architectural style(s)
Software Implementation

- Low level
- Highly precise
- Meaningful to a narrow range of human stakeholders
  - Amenable to manipulation by software tools
- Rich in detail
- “Cloak” architectural information
  - Components, interaction mechanisms, topology, properties
  - Reflect employed architectural styles
Coupling Requirements and Implementation
Example Application: ShareDraw
Architectural Discovery

- Mapping requirements to 7 architectural concerns
  - Data component
  - Processing component
  - Connector
  - Topology (configuration)
  - Component property
  - Connector property
  - (sub)System property

- Results in DAM

- Incremental process
  - Involves technical stakeholders
  - Supports conflict resolution
“DAM-ing” ShareDraw’s Requirements – Components

- Requirement
  - The system should allow the user to directly manipulate graphical objects

is mapped to

- Processing component DAM element
  - Graphical object manipulation component
- Data component DAM element
  - Data for abstract depiction of graphical object
“DAM-ing” ShareDraw’s Requirements – Configuration

- Requirement
  - Allow independent customization of application look-and-feel and graphical object manipulation tools

is mapped to

- Configuration DAM element
  - Strict separation of graphical object manipulation, visualization, and storage components
“DAM-ing” ShareDraw’s Requirements – Property

- Requirement
  - The user should be able to view the effects of his actions with minimal perceived latency

is mapped to

- Component property DAM element
  - Graphical object manipulation component should be efficient, supporting incremental updates
Architectural Recovery

- Four-step process
  - Generate class diagram from implementation
  - Group related classes
    - Isolated classes
    - Aggregation, generalization, composition
    - Two-way associations
  - Package grouped classes into architectural elements
  - Determine partial system configuration
- Results in RAM
Focusing Architectural Recovery

Logical Architecture

- Propose idealized architectural model
- Identify key use cases
- Map components onto architecture
- Generate Refined Architecture

Physical Architecture

- Identify Components
- Analyze component interactions

Diagram:

- Logical Architecture and Physical Architecture are connected through the process steps.
“RAM-ing” ShareDraw’s Implementation – Recovered Application Class Diagram
“RAM-ing” ShareDraw’s Implementation – Isolated Classes
“RAM-ing” ShareDraw’s Implementation – Generalization, Aggregation, Composition
“RAM-ing” ShareDraw’s Implementation – Two-Way Associations
“RAM-ing” ShareDraw’s Implementation – Recovered Partial Configuration

- Summary_Mgr
- FrameWindows_Mgr
- DrawDocument_Mgr
- DrawObj_Mgr
- Dialog_Mgr
- DrawTool_Mgr
- View_Mgr
- Server

+m_pSummInfo
Reconciling DAM and RAM

- Complete the picture with styles
- Classify styles along seven dimensions
  - Data
  - Structure
  - Behavior
  - Interaction
  - Topology
  - Key properties
  - Domain(s) of applicability
- Select appropriate style(s)
- Apply style(s) to DAM to complete the architecture
- Map RAM onto this architecture
## Example Style Classification

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>Structure</th>
<th>Topology</th>
<th>Behavior</th>
<th>Interaction</th>
<th>Properties</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C2</strong></td>
<td>Discrete events</td>
<td>Separable components</td>
<td>Limited component dependencies</td>
<td>Exposed via named services only</td>
<td>Asynchronous coordination</td>
<td>Distributability</td>
<td>GUI Systems</td>
</tr>
<tr>
<td></td>
<td>Data tuples</td>
<td>Explicit connectors</td>
<td>Partially ordered connectivity-based “top” and “bottom” relations</td>
<td>Data queueing and buffering by connectors</td>
<td>Implicit invocation</td>
<td>Heterogeneity</td>
<td>Composability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dynamic creation of connections</td>
<td>Multi-tasking mechanisms such as threads</td>
<td>Event-based interaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Client-server</strong></td>
<td>Parameterized request</td>
<td>Independent servers</td>
<td>Many-to-many connections among clients and servers</td>
<td>Listening server</td>
<td>Server location</td>
<td>Distributability</td>
<td>Distributed Systems</td>
</tr>
<tr>
<td></td>
<td>Typed response</td>
<td>Specialized clients</td>
<td>Dynamic creation of connections</td>
<td>Connections setup and teardown</td>
<td>Remote connection and communication protocol</td>
<td>Security</td>
<td>Evolvability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distributed protocol stacks</td>
<td></td>
<td>Buffering and queuing of requests</td>
<td>Implicit server invocation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Multi-tasking mechanisms such as threads</td>
<td>Data marshalling and unmarshalling</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Exposed via named services only</td>
<td>Client call synchronization</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pipe-and-filter</strong></td>
<td>Streams of typed records</td>
<td>Explicit pipes and filters</td>
<td>Stream between a pipe and a filter</td>
<td>Stream transformation state machine</td>
<td>Synchronization between filter reads and writes</td>
<td>Heterogeneity</td>
<td>Reusability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Input and output ports on filters</td>
<td>No two sources or sinks connected to the same port instance</td>
<td>Data buffering by pipes</td>
<td>Propagation of stream contents to sinks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sources and sinks on pipes</td>
<td></td>
<td></td>
<td>Data marshalling and unmarshalling</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Push-based</strong></td>
<td>Channel notification</td>
<td>Independent producers</td>
<td>Producers connected only to distributors</td>
<td>Content filtering in distributors</td>
<td>Distributor location</td>
<td>Distributability</td>
<td>Distributed Systems</td>
</tr>
<tr>
<td></td>
<td>Subscription request</td>
<td>Explicit distributors</td>
<td></td>
<td>Buffering and queuing by distributors</td>
<td>Remote connection and communication protocol</td>
<td>Scaleability</td>
<td>Robustness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel access/subscribers</td>
<td>Many-to-many channels among receivers and distributors</td>
<td>Subscription setup</td>
<td>Data marshalling and unmarshalling</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Receiver user interface</td>
<td></td>
<td>Content storage/expiration</td>
<td>Distribution policy</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Implicit invocation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ShareDraw’s Architecture – Client-Server + C2
Discussion

- Leverage all available information
  - Requirements can/should play a role in stemming erosion
  - Styles are key
- Do not assume that anything is correct
- Architectural discovery leverages CBSP
- Architectural recovery leverages Focus
- Other techniques possible
- Many interesting questions remain unanswered