Rapid Development of Custom Software Architecture Design Environments

Robert T. Monroe
Carnegie Mellon University
Introduction and Motivation

- Introduction and motivation
- Capturing design expertise
- Customizing design environments
- Validation
- Wrapup
Software Architecture

Software architecture design focuses on:
- Decomposing a system into components
- Interactions between those components
- Emergent global system properties
Premises

- Software Architects can benefit from powerful design environments
  - CAD in other engineering disciplines
  - Design analysis, guidance, and reuse

- The more closely a tool matches the problem it addresses, the more leverage that tool provides
  - Hammer ↔ Nail, Screwdriver ↔ Screw
Example Environment: C2
Example Environment: Meta-H
Example Environment: Aesop/PF
Example Environment: ObjecTime

- Package
  - atmAadSimDemo
  - InterimLmi(v 1.2)
  - AtmAccessDevice(v 1.2)
  - AtmAdapLayer
  - AtmAadSim
  - AtmLayer

- Actor Class
  - CallController(v 1.3)
  - CallControl(v 1.3)
  - Dss1Message(v 1.2)
  - SignallingQ931(v 1.3)
  - SignallingQ2931U(v 1.4)

- Protocol Class
  - CallControlProtocol(v 1.3)
  - Dss1Protocol(v 1.2)
  - Q931*CallControl(v 1.3)
  - Q931*Protocol(v 1.3)

- Data Class
  - T803(v 1.2)
Problems

- Building custom design environments is:
  - Expensive
  - Time consuming
  - Difficult

- Designers’ tooling needs change as their understanding of the problem, domain, and target system evolves
Solution: *Armani*

- Support lightweight, incremental adaptation and customization of design environments
  - Factor out common infrastructure
  - Capture variable design expertise
  - Configure infrastructure with expertise
## Unclogging Bottlenecks

<table>
<thead>
<tr>
<th>Task</th>
<th>Armani changes</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain analysis</td>
<td>- Still requires domain understanding</td>
<td>Small</td>
</tr>
<tr>
<td></td>
<td>- …but provides expressive structure</td>
<td></td>
</tr>
<tr>
<td>Creating schema and capturing expertise</td>
<td>- Armani defines schema and provides structure for capturing expertise declaratively</td>
<td>Large</td>
</tr>
<tr>
<td>(Re-) design, implement, integrate tools, and test environment</td>
<td>- Core infrastructure reused</td>
<td>Huge</td>
</tr>
<tr>
<td></td>
<td>- Basic tooling generated from expertise description.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Still must implement rich tooling</td>
<td></td>
</tr>
<tr>
<td>Adapt and evolve environment</td>
<td>- Configurability and modularity greatly reduces evolution difficulty</td>
<td>Large</td>
</tr>
</tbody>
</table>
Claim 1:
– It is possible to capture software architecture design expertise with a language and mechanisms for expressing *design vocabulary*, *design rules*, and *architectural styles*.

Claim 2:
– This captured design expertise can be used to rapidly and incrementally customize software architecture design environments.
Capturing Design Expertise

- Introduction and motivation
- Capturing design expertise
- Customizing design environments
- Validation
- Wrapup
Architectural Design Expertise

The concepts, models, and rules that skilled architects use when specifying, constructing, or analyzing a software architecture.

Armani provides a declarative language for capturing architecture design expertise.
Capturing Design Expertise

- Design vocabulary
  - Building blocks for system designs
  - e.g. client, web-server, database, pipe, RPC

- Design rules
  - Invariants, heuristics, and analyses
  - e.g. “Transaction rate must be $\geq 1000$ tph”

- Architectural styles
  - Package related vocabulary and design rules
  - e.g. Client-server, pipe-filter, batch sequential
Component Type naïve-client = {

Port Request = {
    Property protocol = rpc-client
};

Property request-rate : integer
    << default = 0; units = “rate-per-sec” >>;

Invariant forall p in self.Ports |
    (p.protocol = rpc-client);
Invariant size(Ports) <= 5;
Invariant request-rate >= 0;
Heuristic request-rate <= 100;
}

**Design Rule Example**

```plaintext
System simpleCS = { ... 
    // simple rule requiring a primary server  
    Invariant exists c : server in self.components |  
        c.isPrimaryServer == true;

    // simple performance heuristic  
    Heuristic forall s : server in self.components |  
        s.transactionRate >= 100;

    // do not allow client-client connections  
    Analysis no-peer-connections(sys : System) : boolean=  
        forall c1, c2 in sys.components |   
            connected(c1, c2) ->  
            !(declaresType(c1,clientT)  
                and declaresType(c2, clientT));

... };
```
### Architectural Style Example

```plaintext
Style naïve-client-server-style = {
    // declare vocabulary
    Component Type naïve-client = {...};
    Component Type naïve-server = {...};
    ...

    // declare design analyses
    Analysis no-peer-connections(sys : System)
        : boolean = { ... };
    ...

    // declare style-wide design rules
    Invariant no-peer-connections(self);
    Heuristic forall s : server in self.components |
        s.transactionRate >= 100;
    ...
}
```
**Predicate-Based Expertise Capture**

- (Most) expertise represented w/predicates
  - Simple type checking tests constraints
  - Predicates can be written over structure, properties, topology
- Clean, flexible approach to subtyping
- Excellent compositionality and modularity
- Predicates can apply to types *or* instances
Language Supports Approach

- Language provides environment foundation
  - Good representations ease environment impl.
  - Reconfigures environment “on the fly”
- Language provides flexible representation for
  - Types
  - Design rules
  - Design instances
- Constraint checking forms tool foundation
Customizing Design Environments

- Introduction and motivation
- Capturing Architecture Design Expertise
- Customizing Design Environments
- Validation
- Wrapup
Example Environment: Armani
Customizing Design Environments

- **Fundamental approach**
  - Provide a fully-functional generic environment
  - Support fine- and coarse-grained customization

- **Key design goals**
  - Incremental effort leads to incremental payoff
  - Standard, common customizations are quick, easy, and incremental.
  - More complex customizations are possible
  - Leverage design language as much as possible
Core Environment Infrastructure

Generic elements:
- Design Rep w/API
- Parser & unparsers
- Type checker
- Analysis Engine
- GUI

Java VM

Visio-based GUI (external tool)
Generic Armani Environment

**Capabilities:**
- Define arch. specifications
- Brings some rigor to box-and-line drawings

**Limitations**
- Limited semantics
- Architect must build-up design concepts
Customization Techniques

- **Fine-grained**
  - Add or modify envt’s stored design expertise
  - Customize graphical depictions within a GUI

- **Coarse-grained**
  - Integrate external tools
  - Completely replace GUI
Fine-Grained Customization

Generic Infrastructure + Custom Environment

Add To or Retrieve From Design Expertise Repository

Define or Modify Expertise

Define or Modify Visualizations

(repeat)
Coarse-Grained Customization

- Coarse-Grained \( \approx \) external tool integration
- Some expertise is better captured with tools
  - When it *does something* to or with a design
  - When it is contained in legacy tools
  - When you have to specify *how* to evaluate it instead of just *what* to evaluate
- More effort, (potentially) more power
Integrated Tool Examples

- Multiple Armani user interfaces
- Performance analysis tool (Lockheed study)
- Change impact and configuration consistency analysis tools (MetaS study)
- Security and fault-tolerance analysis tool (DesignExpert study)
- Runtime architecture evolution checking (C2 study)
Integrating External Tools: UIs

- Command Line Interpreter
  - Scriptable textual interpreter
  - Integrated with direct Armani API calls
  - Simple procedure-call connector, same process
Integrating External Tools: UIs

- **AcmeStudio GUI**
  - Acme design environment front-end
  - Integrated via Acme text stream connector
  - Transport protocol encapsulated in connector
Integrating External Tools: UIs

- Visio-based Armani GUI
  - Highly configurable COTS-based front-end
  - Integrated using sophisticated COM-based intf.
  - Workshops generated by “factory” in connector
Design Environment Conclusions

- Environment demonstrates feasibility of configuring tools with design expertise
- Case studies will demonstrate utility...

- Environment leverages design language
- Support for both fine- and coarse-grained customization was critical
Validation

- Introduction and motivation
- Capturing Architecture Design Expertise
- Customizing Design Environments
- Validation
- Wrapup
Thesis

- Claim 1:
  - It is possible to capture software architecture design expertise with a language and mechanisms for expressing *design vocabulary, design rules, and architectural styles*.

- Claim 2:
  - This captured design expertise can be used to rapidly and incrementally customize software architecture design environments.
Experimental Structure

- Basic approach: proof by existence
- Step 1: task analysis
  - Establish a baseline and find current bottlenecks
- Step 2: build multiple Armani environments
  - Demonstrate breadth, power, and incrementality
- Step 3: external case studies
  - Determine if others can use this technique
Step 1: Task Analysis Findings

- Current techniques require months or years of effort to build a custom environment

- Currently, most development time and effort is devoted to (re)building infrastructure

- If adaptability is not built in from the beginning, evolving an environment can be very difficult
## Step 1: Task Analysis

<table>
<thead>
<tr>
<th>Task</th>
<th>Approximate Time Required (in Engineer/Days, Weeks, Months, or Years)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best Case</td>
</tr>
<tr>
<td></td>
<td>Traditional</td>
</tr>
<tr>
<td>(1) Domain Analysis</td>
<td>Week</td>
</tr>
<tr>
<td>(2) Schema Capture</td>
<td>Days</td>
</tr>
<tr>
<td>(3) Design, implement, test and deploy environment</td>
<td>Months</td>
</tr>
<tr>
<td>Cumulative time to initial deployment</td>
<td>Months</td>
</tr>
<tr>
<td>(4) Time required for environment updates and modifications.</td>
<td>Hours or Days</td>
</tr>
</tbody>
</table>
Step 2: Build Test Environments

- Demonstrate breadth, power, incrementality
  - **Breadth**: build environments for diverse styles
  - **Power**: add significant design expertise to the environments
  - **Incrementality**: adapt and extend environments

- Style selection
  - All case studies based on published style specs
  - At least one from each “Boxology” category
## Step 2: Environments Built

<table>
<thead>
<tr>
<th>Base Style</th>
<th>Specific Style</th>
<th>Total Hours of Effort</th>
<th>Types Defined</th>
<th>Design Rules Defined</th>
<th>Shapes Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Call and Return</strong></td>
<td>Base Driver-Subprogram</td>
<td>6.50</td>
<td>18</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Driver-Subprogram w/DB</td>
<td>4.00</td>
<td>12</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td><strong>Data-Centric</strong></td>
<td>Naïve client-server</td>
<td>2.75</td>
<td>7</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td><strong>Repositories</strong></td>
<td>Three-Tier client-server</td>
<td>3.50</td>
<td>9</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td><strong>Hierarchical</strong></td>
<td>Layered</td>
<td>7.25</td>
<td>11</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td><strong>Data Flow</strong></td>
<td>Batch Sequential</td>
<td>9.25</td>
<td>29</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td><strong>Data Sharing</strong></td>
<td>Armani Design Env.</td>
<td>4.50 (*)</td>
<td>39</td>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td><strong>Interacting</strong></td>
<td>C2 rebuild in Armani</td>
<td>8.00</td>
<td>39</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td><strong>Processes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 2: Driver-Subprogram Style

**Driver-Subprogram Style**

**Category:** Call and Return

**Semantics Statistics**

**Primary component types:**
- Driver
- Subprogram
- Subdriver

**Primary connector type:**
- Processing Request

**Sample design rule:**
- A system has exactly one toplevel Driver component, but may have multiple Subdriver components.

**New types defined:** 18

**Style-wide design rules:** 3

**Time to define:** 3.5 hours

**Lines of Armani code:** 73

**Environment Statistics**

**New shapes defined:** 7

**Customization time:** 3 hours
Step 2: DB-Driver Subprogram Style

**DB-Driver-Subprogram Style**

**Category:** Call and Return

**Semantics Statistics**

**Extended component types:**
- Transaction Manager
- Database
- DB Access SubProgram

**Extended connector types:**
- DB Query Update
- Transaction Request

**Sample design rule:**
- A system has exactly one Transaction Manager that must be connected to all databases.

**New types defined:** 12

**Additional design rules:** 6

**Time to define:** 2.0 hours

**Lines of Armani code:** 63

**Environment Statistics**

**New shapes defined:** 12

**Customization time:** 2 hours
Step 3: External Case Studies

- Qualitative “external” case studies asked:
  - Can other people use Armani effectively?
  - Powerful design expertise capture capabilities?
  - What aspects of Armani worked well?
  - What worked poorly?

- Case study selection criteria
  - Real architects or developers
  - Solving a real problem
Step 3: External Case Studies

Conducted four external case studies:

- SEI’s MetaS architectural style project
  » Change impact and configuration consistency analysis
- UC Irvine C2-integration
  » Run-time architecture evolution analysis
- Lockheed Martin/EDCS
  » Built environment to model and analyze GTN
  » Integrated performance analysis tool
- KeySoftware’s “DesignExpert” tool
  » Developed analyses for reliability and fault-tolerance
Step 3: Case Study Observations

- Can other people use Armani effectively?
  - Yes.

- Powerful design expertise capture?
  - Yes.
  - Case studies spanned broad variety of expertise
  - Case study tools solved real design problems
Step 3: Case Study Observations

◆ What aspects of Armani worked well?
  – Core concepts are flexible and powerful
  – Design representation and checking infrastructure more valuable than GUI

◆ What did not work so well?
  – Declarative design language requires reorientation of thought process
  – Building complex analysis and generation tooling still requires significant effort
Wrapup

- Introduction and motivation
- Capturing Architecture Design Expertise
- Customizing Design Environments
- Validation
- Wrapup
Contributions

- **A technique** for rapidly developing custom software architecture design environments
- **A design language** that captures both design expertise and architectural instances
- **A reference architecture** for highly configurable design environments
- **A set of case studies** that illustrate how to use the technique, language, and environment effectively
Related Work

- Aesop and Acme
- Architecture Description Languages (ADLs)
- Configurable programming environments
  - esp. Gandalf and The Synthesizer Generator
- Design patterns
- Formal specification languages (esp. PVS)
- Constraint-based prog. tools and languages
- DSSA
Future Work

- Generalized reconfiguration strategies
- Integration with full lifecycle processes
- Guidance in selecting styles and expertise
- Discovering new uses for the tools
Conclusions

- The Armani approach to capturing design expertise and incrementally configuring design environments works.

- The Armani conceptual framework can capture a significant range of interesting architectural design expertise.

- Predicate types are a useful abstraction for capturing and composing design expertise.
The End

Robert T. Monroe
Carnegie Mellon University
Type FastT = {
  Prop. latency = ...
  Prop. throughput = ...
  Invariant latency < ...
  Invariant throughput > ...
}

Type DatabaseT = {
  Prop. schema = ...
  Prop. transRate = ...
  Prop. multiThreaded = ...
}

Type TransactionalT = {
  Prop. transProtocol = ...
  Prop. rollbackPolicy = ...
  Invariant (transProtocol != "")
}

Type FastTransDatabaseT = {
  Prop. schema = ...
  Prop. transRate = ...
  Prop. multiThreaded = ...
  Prop. latency = ...
  Prop. transProtocol = ...
  ...
}
Standard Customization Process

- Load design expertise captured with Armani design language into generic envt.
- Create custom icons to represent new design elements (optional)
- Modify expertise as needed
- Repeat
Tools manipulate Armani designs through a programmatic API. (In or out of Java VM)
Armani Environment Architecture

Java-Based Workshops

Textual Armani Description

Armani Parser

Armani Exporter

Java VM

COM Methods

Dispatch

GUI InterfaceClass

Armani Design Representation

"Linked" Armani Tool (or intf)

Type Manager/Constraint Checker

Core Java Representation

"Pure-Java" Design Representation

EditMe()

UpdateMe()

UpdateMe()

UpdateMe()

EditMe()

EditMe()
The generic Armani environment provides:
- API for manipulating design representation
- Parser and unparsener for design language
- GUI
- Design checker
- Tool integration framework
Customizing Visualizations

- Different types of vocabulary elements require different visualizations
- Visio™ used as the generic GUI front end, handles visualization specialization
- GUI Front-end is just another tool
  - It can be exchanged for a different front-end
  - Visualizations are highly independent of underlying semantic representation
Customizing Visualizations

Challenge: Visualization semantics don’t work compose like architectural semantics

Solution:
- Associate visualizations with “templates” or “macros” instead of types.
## Task Analysis - State of Practice

<table>
<thead>
<tr>
<th>Task</th>
<th>Best Case</th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Domain Analysis</td>
<td>Week</td>
<td>Month(s)</td>
<td>Year(s)</td>
</tr>
<tr>
<td>(2) Schema Capture</td>
<td>Days</td>
<td>Weeks</td>
<td>Months</td>
</tr>
<tr>
<td>(3) Design, implement, test and deploy environment</td>
<td>Month(s)</td>
<td>Months or Years</td>
<td>Years or until project cancellation</td>
</tr>
<tr>
<td>Cumulative time to initial deployment</td>
<td>Months</td>
<td>Months or Years</td>
<td>Years or until project cancellation</td>
</tr>
<tr>
<td>(4) Time required for environment updates and modifications.</td>
<td>Hours or Days</td>
<td>Months</td>
<td>Months</td>
</tr>
</tbody>
</table>
Creating a Design Environment

- Creating a custom environment requires ...
  - Domain analysis
  - Create schema for designs and design expertise
  - Design, implement, test, and deploy envt.
  - Modify and evolve environment as needed
Integrating External Tools: UIs

- Armani UI implemented as external tool
- Three integration connector types provided:
  - *Direct Java API call* for Java-based tools that run in the same process space as the Armani core infrastructure
  - *Acme text stream* for Acme-compliant tools
  - *Custom COM interfaces* for arbitrary external tool integration.

  » Builds tool-specific, semantically rich, interfaces on top of the generic Armani Java interfaces