On Verifying the Use of a Pattern Language in Model Driven Design

Bahman Zamani

Department of Computer Engineering
Faculty of Engineering
University of Isfahan

June 6th, 2010
1. Introduction
   - Research Area
   - Problem & Solution
   - Background
   - Related Work

2. Pattern Language Verification
   - Simple PLV Process
   - Describing a Pattern Language
   - Motivating Example
   - Rule Formalisms
   - Profile-Driven PLV Process
   - The Case Study

3. Conclusion
   - Contributions
   - Limitations
   - Future Work
1. Introduction
   - Research Area
   - Problem & Solution
   - Background
   - Related Work

2. Pattern Language Verification
   - Simple PLV Process
   - Describing a Pattern Language
   - Motivating Example
   - Rule Formalisms
   - Profile-Driven PLV Process
   - The Case Study

3. Conclusion
   - Contributions
   - Limitations
   - Future Work
Problem & Solution

Problem: Automating the use of a Pattern Language in a design model and checking that the patterns are applied appropriately.

Sub-Problems:
1. What is a Pattern Language?
2. Describing a Pattern Language
3. Formalizing a Pattern Language
4. Extracting formalized rules from textual description of patterns
5. Providing tool support for usage of PL in model driven design
6. Pattern Selection
7. Code Generation
8. Building tools automatically from the formalized Pattern Language

Solutions:
- A definition is selected
- Structure, Syntax, and Semantics
- Formalisms are defined
- An approach is presented via the PofEAA case study
- ArgoPLV tool is developed in the case study
- Pattern Selection
- Code Generation
- Building tools automatically from the formalized Pattern Language

<table>
<thead>
<tr>
<th>Author/Year</th>
<th>Title</th>
<th>Publication</th>
<th>Source/Model</th>
<th>Pattern/PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lovatt 2005</td>
<td>A Pattern Enforcing Compiler (PEC) for Java</td>
<td>PhD Thesis, Macquarie Univ., Australia</td>
<td>Source</td>
<td>Pattern</td>
</tr>
<tr>
<td>Busch et. al 2007</td>
<td>Pattern-Oriented Software Architecture, v. 5</td>
<td>John Wiley</td>
<td>Model</td>
<td>PL</td>
</tr>
<tr>
<td>Zdun 2007</td>
<td>Systematic pattern selection using pattern language grammars and design space analysis</td>
<td>SPE, v. 37</td>
<td>N/A</td>
<td>PL</td>
</tr>
</tbody>
</table>
• Paradigm shift in software development
  – From code-centric to model-centric
  – Models are first class citizens
• The ultimate goal is to automatically generate programs from the corresponding models [Sel 03].
• Model-Driven Approaches of Software Development:
  – Model-Driven Development (MDD)
  – Model-Driven Architecture (MDA)
  – Model-Driven Engineering (MDE)
  – Model-Driven Software Development (MDSD)
  – Model-Driven Software Engineering (MDSE)
• They are all “Model-Driven”
• Automatic (full) code generation
  – Forward Engineering
  – Backward Engineering

Background: An MDE Road Map

- MDE is a model-centric approach.
- Software development is correct application of some transformations.
- Developers transform artifacts from one level of abstraction to another level, until they obtain a working code.
- For applying MDE, we need a modeling tool to utilize automatic execution of transformations.
- Patterns are part of the designer’s domain knowledge.
1. Introduction

2. Pattern Language Verification

3. Conclusion

Background: Patterns & Pattern Languages

- Pattern Languages in Architecture:
  - “Pattern” and “Pattern Language” coined in late 60’s by Christopher Alexander an emeritus professor of architecture at the University of California at Berkeley.
  - Alexander’s books:
    - “A Pattern Language: Towns, Buildings, Construction” (1977)
    - “The Timeless Way of Building” (1979)
  - In 2000, Alexander founded the patternlanguage.com company to promote collaborative working between people, builders, and architects to build good buildings.
  - Sitting Wall Pattern →
An excerpt from patternlanguage.com which summarizes the Pattern Language concept:

Once upon a time, we wrote a book called A Pattern Language and that is how we got our name. [...] The new idea in the book was to organize implicit knowledge about how people solve recurring problems when they go about building things. [...] Patterns are easy to remember and set out as if-then propositions. [...] We were surprised though, when we found out computer programmers liked it, because it was about building not programming. But the programmers said, “this is great, it helps think about patterns in programming and how to write reusable code that we can call upon when we need it.”

1. Introduction

2. Pattern Language Verification

3. Conclusion

Background: Patterns & Pattern Languages (in Software)

Take Home Message:

“Patterns are Important!”

“Pattern Languages are not well developed!”

- [SFP 09] http://cse-ferg41.unl.edu/SFP
# Background: Short History of Software Patterns

<table>
<thead>
<tr>
<th>Year</th>
<th>Patterns</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>1987</td>
<td>First software patterns: Five patterns for designing Smalltalk windows [Beck 87]</td>
<td>Source Code</td>
</tr>
<tr>
<td>1994</td>
<td>First conference of “Pattern Language of Programs” (PLoP) [PLoP 94]</td>
<td>Pattern Languages</td>
</tr>
<tr>
<td>1997</td>
<td>Martin Fowler’s book “Analysis Patterns: Reusable Object Models” [Fowler 97]</td>
<td>Analysis Patterns</td>
</tr>
<tr>
<td>2002</td>
<td>Martin Fowler’s book “Patterns of Enterprise Application Architecture” (PofEAA) [Fowler 02]</td>
<td>Architectural Patterns</td>
</tr>
<tr>
<td>2007</td>
<td>Vol. 5 of book “Pattern–Oriented Software Architecture” (POSA) [PoSA 07]</td>
<td>Pattern Languages</td>
</tr>
</tbody>
</table>

- [PLoP 94] [http://www.plop.org](http://www.plop.org)
GoF Design Patterns Book [Gam 95]:
- Bible of design patterns
- Citation count on ACM is 1648
- Citation count on Google Scholar is 14608
- With 243 reviewers in amazon.com, it ranks 2\textsuperscript{nd} in the Software Engineering Bestsellers category

[Note: Numbers captured on July 2007]

- Example: Observer Design Pattern
- Problem: Update interfaces based on the changes
- Solution: the observer observes the changes and updates its display.

\[\text{Gam 95}\] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
• Pattern Relationships:
  – Patterns are not isolated islands.
  – Example: Relationships Between GOF Design Patterns Proposed in [Zim95]

1. Introduction

2. Pattern Language Verification

3. Conclusion

Background: Patterns of Enterprise Application Architecture (PofEAA)

- 51 patterns for designing web-based enterprise applications
- Layered architecture
  - Main layers
    - Presentation
    - Domain
    - Data Source
  - Supporting layers
- Our Case Study: 23 patterns

<table>
<thead>
<tr>
<th>Layer</th>
<th>Patterns</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Domain</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Data Source</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>O/R Mapping</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Distribution</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Concurrency</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Session</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Base</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>51</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>
Related Work

1. Introduction

2. Pattern Language Verification

3. Conclusion

Take Home Message:

“No work on Pattern Languages in Models!”

Overview

1. Introduction
   - Research Area
   - Problem & Solution
   - Background
   - Related Work

2. Pattern Language Verification
   - Simple PLV Process
   - Describing a Pattern Language
   - Motivating Example
   - Rule Formalisms
   - Profile-Driven PLV Process
   - The Case Study

3. Conclusion
   - Contributions
   - Limitations
   - Future Work
The PLV process is inspired by the compilers:
- PSV verifies the Structural rules
- PTV verifies the Syntactic rules
- PMV verifies the Semantic rules
Three aspects of a pattern language:

1. **Structural**: Defines the structure of individual patterns
   
   e.g., Front Controller pattern needs a Handler and a Command Structure, …

2. **Syntactic**: Defines the organization and the relationship between patterns
   
   – Pattern-Layer Relationships
     
     e.g., Data Transfer Object pattern is located in the Distributed layer
   
   – Pattern-Pattern Relationships
     
     e.g., Simple Domain Model uses Active Record pattern.

3. **Semantic**: Defines the consistency with context information
   
   e.g., Template View pattern is for HTML pages.
Motivating Example: Online Student Registration System

1. Introduction

2. Pattern Language Verification

3. Conclusion

Context Information
- Complexity: High
- ViewBuilt: XSLT

Semantic Error
"Inconsistency with context information: ViewBuilt=XSLT"

Structural Error
"Missing doGet() and doPost() operations"

Syntactic Error
"Relationship between DomainModel and ActiveRecord Pattern conflicts with the Complexity=High"

Syntactic Error
"Not in the appropriate layer"

04/06/2010 19/36

Pattern Language Verification, Bahman Zamani
**Formalism**: A Combination of textural rules in plain English with UML class diagrams.

**Example**: The Front Controller Pattern

1. There is a Front Controller (=Handler) class in the model.
2. There are at least two operations (doGet and doPost) in the Handler class.
3. The Handler class has a client dependency to a Command class.
4. The Command class is abstract.
5. The Command class has at least one process operation.
6. The Command class has at least one Concrete Command child class.
7. A Concrete Command class is concrete.
8. A Concrete Command class has at least one process operation.
Syntactic Rule Formalism (Pattern-Layer Relationships)

Formalism:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l m$</td>
<td>Layer</td>
</tr>
<tr>
<td>$P$</td>
<td>Pattern</td>
</tr>
<tr>
<td>$\supset$</td>
<td>Layer Inclusion</td>
</tr>
<tr>
<td>$\exists$</td>
<td>Pattern Membership</td>
</tr>
<tr>
<td>$,^+$</td>
<td>Group Inclusion</td>
</tr>
<tr>
<td>$,^-$</td>
<td>Group Membership</td>
</tr>
<tr>
<td>$\cdot$</td>
<td>Layer dependency</td>
</tr>
<tr>
<td>$*$</td>
<td>Optional Layer</td>
</tr>
<tr>
<td>$(c)$</td>
<td>Conditional Layer</td>
</tr>
<tr>
<td>${c}$</td>
<td>comment</td>
</tr>
</tbody>
</table>

Example:

```
opoeaa model $\supset$ main layer . auxiliary layer*
main layer $\supset$ presentation . service?($\text{NeedServiceLayer=Yes}$) . domain . datasource
presentation $\supset$ controller . view

controller $\exists$ Page Controller . Front Controller
```
Syntactic Rule Formalism (Pattern-Pattern Relationships)

Formalism:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P )</td>
<td>root pattern</td>
</tr>
<tr>
<td>( \rightarrow )</td>
<td>uses</td>
</tr>
<tr>
<td>( \quad )</td>
<td>conditional uses</td>
</tr>
<tr>
<td>( \mid )</td>
<td>alternative uses</td>
</tr>
<tr>
<td>( \quad )</td>
<td>conflicts</td>
</tr>
<tr>
<td>( \downarrow )</td>
<td>conflicts in layer</td>
</tr>
<tr>
<td>( \uparrow )</td>
<td>refines</td>
</tr>
<tr>
<td>( { c } )</td>
<td>comment</td>
</tr>
</tbody>
</table>

Example:

\[
\begin{align*}
\text{Page Controller} & \rightarrow \text{Template View} \mid \text{Transform View} \\
\text{Page Controller} & \xrightarrow{\text{Tool=Net}} \text{Template View} \\
\text{Template View} & \xrightarrow{\text{NeedServiceLayer=Yes}} \text{Service Layer} \\
\text{Service Layer} & \rightarrow \text{Domain Model} \mid \text{Table Module} \\
\text{Template View} & \xrightarrow{\text{NeedServiceLayer=Yes}} \text{Domain Model} \mid \text{Table Module} \mid \text{Transaction Script} \\
\text{Domain Model} & \xrightarrow{\text{Complexity=Low}} \text{Active Record} \\
\text{Domain Model} & \xrightarrow{\text{Complexity=High}} \text{Data Mapper}
\end{align*}
\]
Semantic Rule Formalism

Formalism:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\approx$</td>
<td>consistent</td>
</tr>
<tr>
<td>$\neq$</td>
<td>inconsistent</td>
</tr>
</tbody>
</table>

Example:

*Page Controller* $\approx \{\text{Tool} = \text{.Net}\}$
*Front Controller* $\approx \{\text{Tool} = \text{Java}\}$
*Template View* $\approx \{\text{ViewBuilt} = \text{HTML}\}$
*Transform View* $\approx \{\text{ViewBuilt} = \text{XSLT}\}$
*Domain Model* $\approx \{\text{Tool} = \text{Java} \text{ and Complexity} = \text{High} \text{ and Expertise} = \text{Expert}\}$
*Transaction Script* $\neq \{\text{Tool} = \text{.Net}\}$
2. Pattern Language Verification

A Profile-Driven PLV Process

Diagram showing the process of Pattern Language Verification (PLV) with steps involving Pattern Language UML Profile (PLP), UML Design Model, Pattern Structural Verifier (PSV), Pattern Language Syntactic Verifier (PTV), Pattern Language Semantic Verifier (PMV), and Pattern Language Advisor (PLA). The process involves checking for errors and design rationale.
The Case Study: Overview

1. The Pattern Language: PofEAA
   - 23 Selected Patterns
   - 74 Extracted Advices
   - Formalized rules: 23 Structural, 44 Syntactic, 20 Semantic

2. The PofEAA UML Profile
   - 63 Stereotypes, 9 Tagged Values, No OCL Constraints

3. The ArgoPLV Tool
   - Integrated into ArgoUML
   - ArgoUML: UML Modeling tool, Open Source, In Java, 0.5 M downloads, Design Critiquing System (Critics & Wizards)
   - ArgoPLV: Extension of ArgoUML via GUI extensions, 50 Critics, 50 Wizards, 5000+ LOC
   - Critics: Hard coding of Structural, Syntactic, and Semantic rules
   - Wizards: Hard coding of the Pattern Language Advisor (PLA)

4. ArgoPLV in Action: An exemplar session of ArgoPLV
   - Online Student Registration System: Interactive Design vs. Model Checking
The Case Study: From Textual Advices to Formalized Rules

<table>
<thead>
<tr>
<th>A#</th>
<th>Type</th>
<th>Description (PofEAA book page#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A18</td>
<td>Structural</td>
<td>“A Table Data Gateway has a simple interface, usually consisting of several find methods to get data from the database and update, insert, and delete methods...The Table Data Gateway is usually stateless.” (p. 144)</td>
</tr>
<tr>
<td>A13</td>
<td>Syntactic</td>
<td>“A simple Domain Model can use Active Record, whereas a rich Domain Model requires Data Mapper.” (p. 117)</td>
</tr>
<tr>
<td>A23</td>
<td>Syntactic/Semantic</td>
<td>“[for presentation layer] Your tooling may well make your choice for you. If you use Visual Studio, the easiest way to go is Page Controller and Template View. If you use Java, you have a choice of Web frameworks to consider. Popular at the moment is Struts, which will lead you to a Front Controller and a Template View.” (p. 99)</td>
</tr>
</tbody>
</table>

Advice A18 ⇒ Formal Structural Rules: We do not check the statelessness of the TDG pattern.

Advice A13 ⇒ Formal Syntactic Rules:

\[
\text{Domain Model} \xrightarrow{\text{Complexity}=\text{Low}} \text{Active Record and Domain Model} \xrightarrow{\text{Complexity}=\text{High}} \text{Data Mapper}
\]

Advice A23 ⇒ Formal Syntactic Rules:

\[
\text{Page Controller} \xrightarrow{\text{Tool}=\text{.Net}} \text{Template View and Front Controller} \xrightarrow{\text{Tool}=\text{Java}} \text{Template View}
\]

Advice A23 ⇒ Formal Semantic Rules:

\[
\text{Page Controller} \approx \{\text{Tool = .Net}\} \text{ and Front Controller} \approx \{\text{Tool = Java}\}
\]
The Case Study: Profile Stereotypes

1. Introduction
2. Pattern Language Verification
3. Conclusion
### The Case Study: Profile Tagged-Values

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Mult.</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceLayer</td>
<td>String</td>
<td>[0..1]</td>
<td>Yes, No</td>
</tr>
<tr>
<td>DistributedLayer</td>
<td>String</td>
<td>[0..1]</td>
<td>Yes, No</td>
</tr>
<tr>
<td>ConcurrencyLayer</td>
<td>String</td>
<td>[0..1]</td>
<td>Yes, No</td>
</tr>
<tr>
<td>SessionStateLayer</td>
<td>String</td>
<td>[0..1]</td>
<td>Yes, No</td>
</tr>
<tr>
<td>ChanceOfConflict</td>
<td>String</td>
<td>[0..1]</td>
<td>Low, High</td>
</tr>
<tr>
<td>ViewBuilt</td>
<td>String</td>
<td>[0..1]</td>
<td>HTML, XSLT</td>
</tr>
<tr>
<td>Tool</td>
<td>String</td>
<td>[0..1]</td>
<td>Java, .Net</td>
</tr>
<tr>
<td>Complexity</td>
<td>String</td>
<td>[0..1]</td>
<td>Simple, Moderate, Complex</td>
</tr>
<tr>
<td>Expertise</td>
<td>String</td>
<td>[0..1]</td>
<td>Novice, Intermediate, Expert</td>
</tr>
</tbody>
</table>
The Case Study: ArgoPLV Architecture
The Case Study: ArgoPLV in Action

1. Introduction

2. Pattern Language Verification

3. Conclusion
The Case Study: ArgoPLV in Action

1. Introduction
2. Pattern Language Verification
3. Conclusion

Pattern Language Verification, Bahman Zamani
Overview

1. Introduction
   - Research Area
   - Problem & Solution
   - Background
   - Related Work

2. Pattern Language Verification
   - Simple PLV Process
   - Describing a Pattern Language
   - Motivating Example
   - Rule Formalisms
   - Profile-Driven PLV Process
   - The Case Study

3. Conclusion
   - Contributions
   - Limitations
   - Future Work
Contributions

1. The Pattern Language Verifier (PLV) process
2. The formalism for representing a pattern language
3. The ArgoPLV Tool
   - The PofEAA advices
   - The PofEAA formalized rules
   - The PofEAA UML Profile
4. An exemplar session of ArgoPLV
1. PLV is a profile-driven process
2. ArgoPLV is hand-coded
3. Pattern selection is not addressed
4. Domain of the design is not considered
5. Semantic aspects is not mature
6. Case study is limited
Future work

1. Generalizing the idea of PLV and the experiences gained in this work towards a framework for "Pattern Language Verification"

2. The PLV process can also be enriched with the idea of systematic pattern selection

3. Inter-collection pattern support

4. Considering dynamic models

5. Automatically generation of PLV from the formalized pattern language

6. Investigating the synthesis (code-generation) part of a compiler
Thank You!

Questions are Welcomed

More Info at: bahmazamani.com