Checking Architectural and Implementation Constraints for Domain-Specific Component Frameworks using Models

Carlos Noguera\textsuperscript{1}  \hspace{1cm} Frédéric Loiret\textsuperscript{2}

\textsuperscript{1}Software Languages Lab (SOFT)  
Vrije Universiteit Brussels, Belgium

\textsuperscript{2}INRIA ADAM Team  
University of Lille & INRIA Lille – Nord Europe

17\textsuperscript{th} of December, 2009
Outline

Domain Specific Components Using Hulotte

Architectural Consistency and Implementation Conformance

A three-tier Approach to Validation
  Implementation Metamodel
  Pivot Metamodel
  Example: Model of Multitasking/Distributed DSC

Conclusion
Domain Specific Component Frameworks

Component-Based Software Engineering

- Decomposition in logical modules
- Relation between modules
- Architectural artifact’s semantics depend on application domain

Domain-Specific Component Frameworks

- Extend architectural artifacts with domain-specific concerns
- Artifacts closer to the problem domain
- Easier to understand
- Domain-specific validation
Example: A DSC Framework for Multitasking/Distributed Applications

<table>
<thead>
<tr>
<th>Distributed</th>
<th>Multitasking</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Distributed components</td>
<td>▶ Active components</td>
</tr>
<tr>
<td>▶ Asynchronous bindings</td>
<td>▶ Sporadic</td>
</tr>
<tr>
<td></td>
<td>▶ Periodic</td>
</tr>
<tr>
<td></td>
<td>▶ Protected components</td>
</tr>
</tbody>
</table>
Example: Multitasking/Distributed Applications

Distributed

- Distributed Action and Reaction Comp.
- Async binding between them

Multitasking

- Two periodic components
- one protected component between them
- One sporadic component
Domain-Specific Component Framework development is ad-hoc.

Hulotte

- MDE domain-specific component framework development
- Based on a generic metamodel.
- Design-level annotations for Domain-Specific Concerns
- Design and runtime platform
Extend Hulotte metamodel with annotations from **Multitasking** and **Distributed** concerns

How to validate architectural consistency and structural implementation conformance?
Multitasking

- No active and protected components
- No cycles between active and protected components
- active component’s implementation should not spawn new threads

Distributed

- All components must be contained in a distributed component
- Async bindings can only target sporadic components
- Methods in async bindings must be of void type
Architectural Consistency and Implementation Conformance

Domain-specific components come with domain-specific constraints

Constraints at two levels

- Architectural level - domain-specific architecture consistency
- Implementation level - domain/implementation specific conformance

Distributed

- All components must be contained in a distributed component
- Async bindings can only target sporadic components
- Methods in async bindings must be of void type

Multitasking

- No active and protected components
- No cycles between active and protected components
- Active component’s implementation should not spawn new threads
A Three-tier approach to Validation of DSC

Augment DSC metamodel with implementation metamodel to express constraints as invariants

- Architecture model to check design consistency (Hulotte)
- Implementation model to check conformance (Java/Fractal runtime)
- Pivot model to map architectural artifacts to implementation (Java annotations)

Constraints

- Architectural and implementation level expressed in an uniform language (OCL)
- Validated over instances extracted from implementation code and architecture description
Implementation metamodel - SpoonEMF

- **CtType**
  - 1 \text{CtClass}
  - \text{methods} \ast

- **CtClass**
  - \text{parameters} 0..1

- **CtInterface**
  - \text{body} 1

- **CtMethod**
  - \text{body} 1

- **CtBlock**
  - \text{parameters} \ast

- **CtParameter**
  - \text{statements} \ast

- **CtStatement**

**SpoonEMF**

- Full Java5 AST modeled using EMF
- Java code 2 EMF discoverer
- EMF 2 Java unparsen
Pivot - Fraclet

Client.java - Fraclet Component

```java
@Component(name = "helloworld.Client")
public class Client {

    @Attribute(value="Hello world")
    private String message;

    @Requires(name="s")
    private Service service;

    // ...
    @Lifecycle(CREATE)
    protected void whenCreated() {
        log.info("...");
    }
}
```

- Annotations to generate Hulotte/Fractal boilerplate code
- Map Java concepts to Hulotte/Fractal concepts
Pivot - Fraclet Metamodel

Obtained from Fraclet annotations using Modelan
Extends SpoonEMF metamodel
Links between annotation metaclasses and SpoonEMF metaclasses
Fraclet-annotated Java code to EMF discoverer
Architecture layer is connected to the implementation layer through the pivot layer
Example: Model for Multitasking/Distributed DSC

- Hulotte instance generated from ADL
- Fraclet and SpoonEMF instances generated from annotated Java code
- Constraints are checked during Hulotte code generation
Example: OCL Constraints

Constraints at architectural level

<table>
<thead>
<tr>
<th>All components must be contained in a distributed component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context: Component</td>
</tr>
<tr>
<td>def: isDistributedN(c: Component) : Boolean</td>
</tr>
<tr>
<td>= UTIL.isDistributed(c) or</td>
</tr>
<tr>
<td>c.SuperComponent-&gt;exists(cs</td>
</tr>
<tr>
<td>inv: isDistributedN(self)</td>
</tr>
</tbody>
</table>

Async bindings can only target sporadic components

<table>
<thead>
<tr>
<th>Async bindings can only target sporadic components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context: Binding</td>
</tr>
<tr>
<td>inv: UTIL.isAsynchronous(self) implies UTIL.isSporadic(self.de...</td>
</tr>
</tbody>
</table>
Example: OCL Constraints

Constraints at implementation level

Methods in `async` bindings must be of `void` type

Context: Active
inv: `self.annotatesSet->forall( b | SpoonUTIL.getFractalInterface(b.source).target.Methods.Type.SimpleName = 'void')`
Conclusion

Contribution

- Model-based approach for Domain-Specific Component validation
- Merging structural models of architecture and implementation
- Unified language for expressing architectural and implementation level constraints
- Prototype tool integrated to the Hulotte Domain-specific component development framework