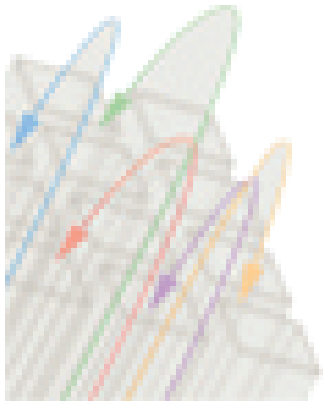
Checking Architectural and Implementation Constraints for Domain-Specific Component Frameworks using Models

Carlos Noguera[1] Frédéric Loiret[2]

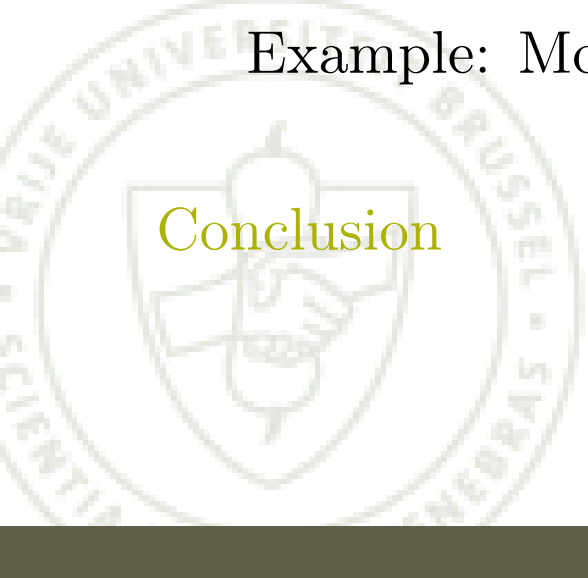
[1]Software Languages Lab (SOFT)
Vrije Universiteit Brussels, Belgium

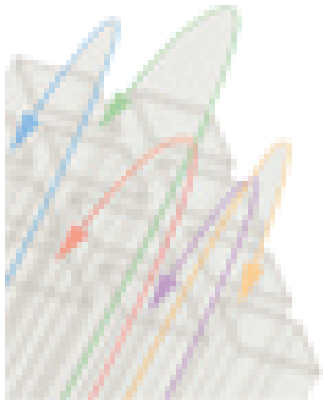
[2]INRIA ADAM Team
University of Lille & INRIA Lille – Nord Europe

17^{th} of December, 2009



Outline





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

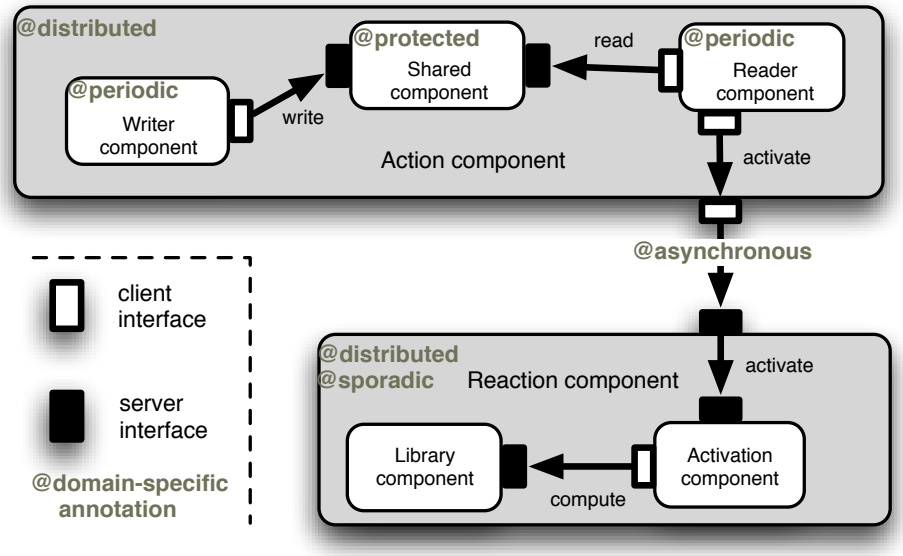
Distributed

- Distributed components
- Asynchronous bindings

Multitasking

- Active components
  - Sporadic
  - Periodic
- Protected components

Example: Multitasking/Distributed Applications



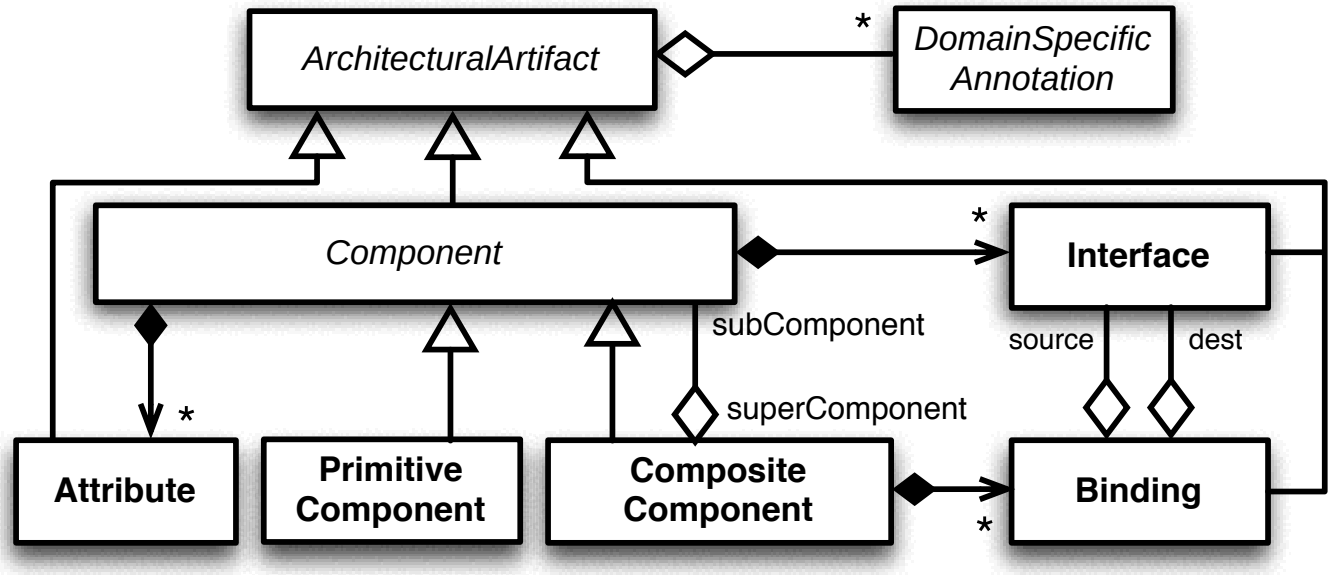
Distributed

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

Multitasking

- Two periodic components
- one protected component between them
- One sporadic component

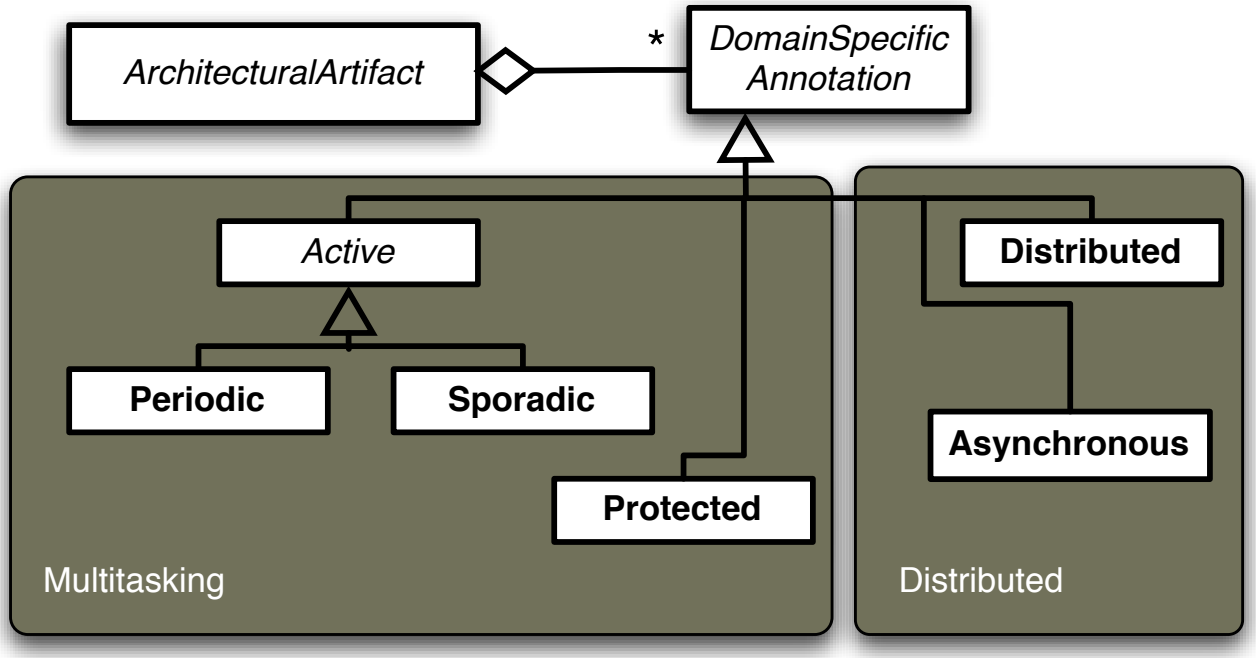
Domain-Specific Component Framework development is ad-hoc.



Hulotte

- MDE domain-specific component framework development
- Based on a generic metamodel.
- Design-level annotations for Domain-Specific Concerns
- Design and runtime platform

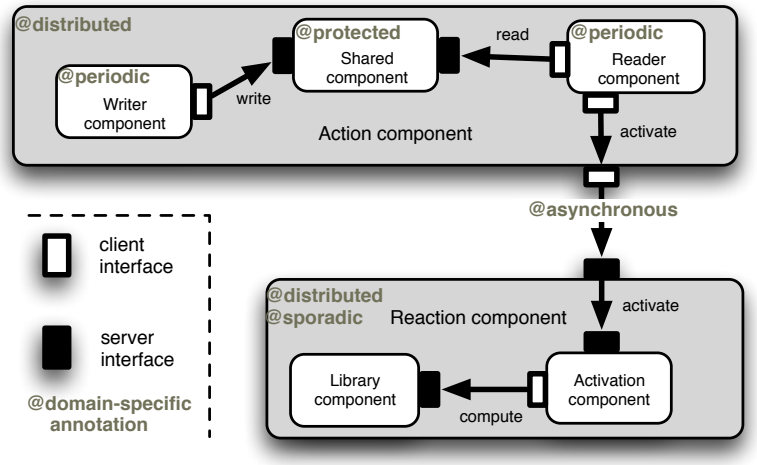
Example: Multitasking/Distributed Hulotte Annotations



Extend Hulotte metamodel with annotations from Multitasking and Distributed concerns

How to validate architectural consistency and structural implementation conformance?

Example: Constraints for Multitasking/Distributed DSC



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

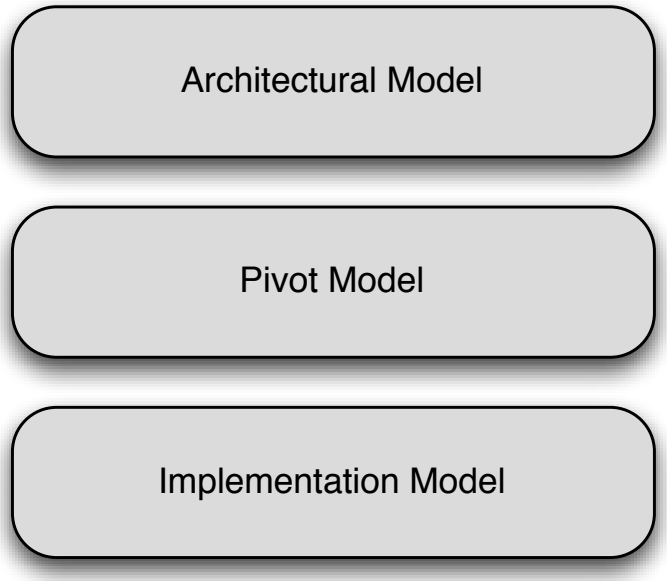
- All components must be contained in a distributed component
- Async bindings can only target sporadic components
- Methods in async bindings must be of void type

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

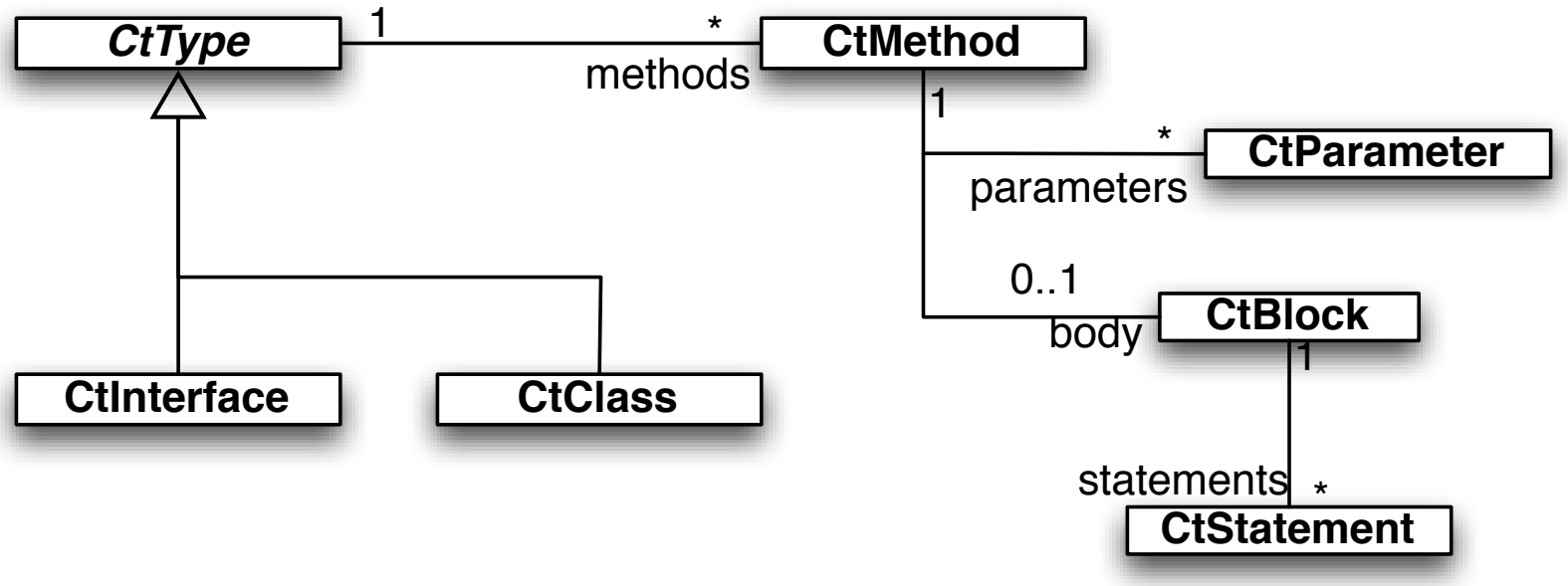


- Architecture model to check design consistency (Hulotte)
- Implementation model to check conformance (Java/Fractal runtime)
- Pivot model to map architectural artifacts to implementation (Java annotations)

Constraints

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

Implementation metamodel - SpoonEMF



SpoonEMF

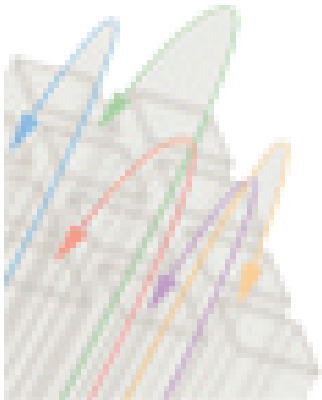
- Full Java5 AST modeled using EMF
- Java code 2 EMF discoverer
- EMF 2 Java unparser

Pivot - Fraclet

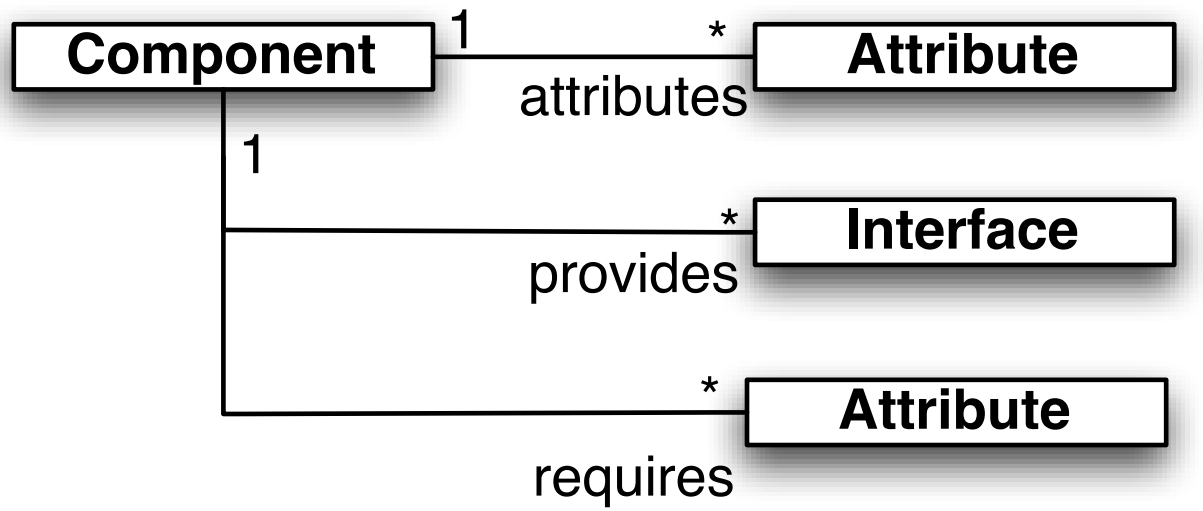
Client.java - Fraclet Component

```
@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



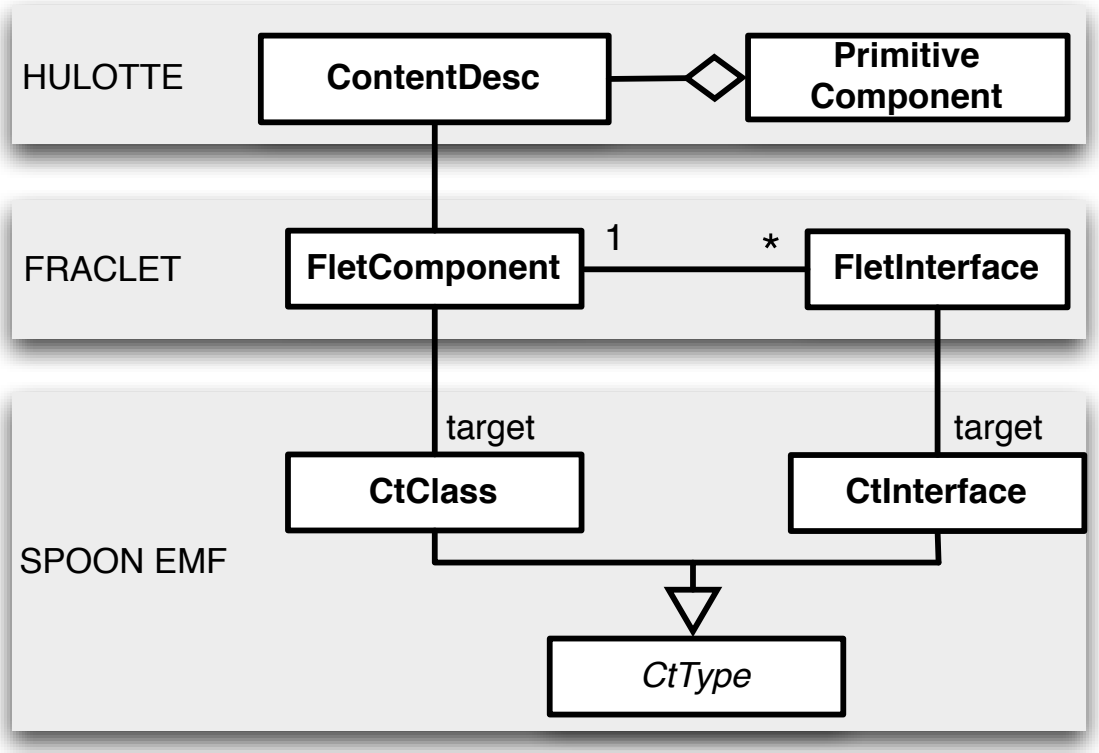
Pivot - Fraclet Metamodel



Fraclet Metamodel

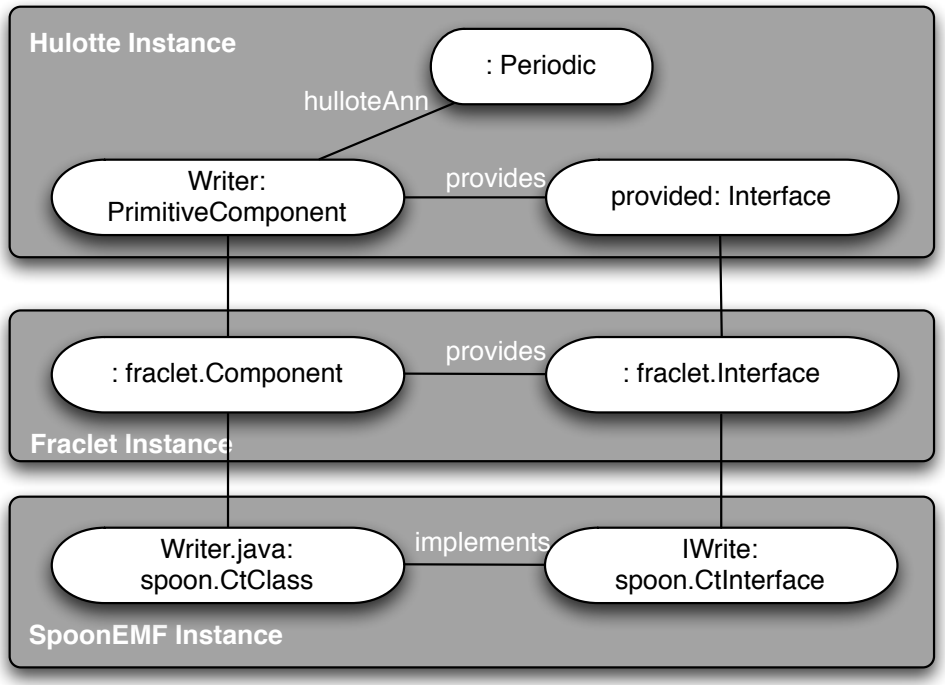
- Obtained from Fraclet annotations using Modelan
- Extends SpoonEMF metamodel
- Links between annotation metaclasses and SpoonEMF metaclasses
- Fraclet-annotated Java code to EMF discoverer

Three-tier DSC metamodel



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

All components must be contained in a distributed component

```
Context:  Component
def:      isDistributedN(c: Component) : Boolean
          = UTIL.isDistributed(c) or
          c.SuperComponent->exists(cs|isDistributedN(cs))
inv:      isDistributedN(self)
```

Async bindings can only target sporadic components

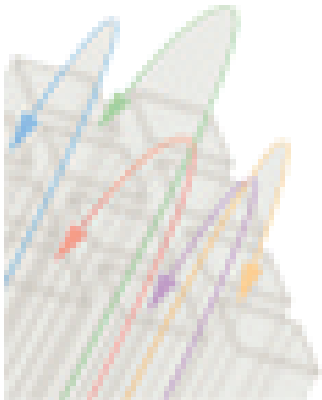
```
Context:  Binding
inv:      UTIL.isAsynchronous(self) implies UTIL.isSporadic(self.d
```

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