



Some Desired Features for the DEVS Architecture Description Language

Olivier Dalle – Judicael Ribault

DEVS/TMS, Boston, April 6th 2011

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE



Presentation Roadmap

- A bunch of questions
- The Fractal Component Model
- Open Simulation Architecture
- (Fractal) ADL for simulation
- Conclusion & Perspectives

A few questions...

What do **We** Need a Standard For ?

•Models?

- Conceptual = we have DEVS (implicitly agreed) + Documentation (ontologies, SES)?
- We miss a standard implementation
 - Some interesting work presented here, DEVSML, XML...
 - What about other/existing implementations?
 - Are they supposed to disappear?
 - How are we going to use the standard?

•Simulations?

- We already have DEVS abstract simulator... (enough?)

•Experiments?

- We already have Experimental Frame...

BTW, Who is 'We' ?

- DEVS Practitioners?
- Experimenters ?
- What about a Simulation standard at large?
 - Reuse/combine DEVS elements and elements from other simulators?
 - HLA ... Coarse grain: reuse of dedicated simulators, loss of structure
 - Reuse more than just simulators and models?
- What other means of combining reusing do we have?
- What else could we try to reuse?

Standardization of Models

Conceptual level

Atomic DEVS =

$$\langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$$

Example:

GEN = $\langle S, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$

- $X = \phi$
- $Y = \{\text{out}\}$
- $S = \{G\}$
- $\delta_{\text{int}}: \delta_{\text{int}}(G) = G$
- $\delta_{\text{ext}}: \text{unavailable}$
- $\lambda(G) = \text{out}$
- $ta(G) = \text{GEN_TIME}$

♪

Implementation Level ??

Atomic DEVS = a derived class ?
which language ?

Example (DEVJava) :

```
public class Gen extends
ViewableAtomic {
    ...
    public void deltint() {
        holdIn("active", period_);
    }
}
```

Is Simulation so Different from Everything Else?

Many component models

- EJB, COM, SCA...
 - Flat structure, eg. Composite / Components / Services (SCA)

Even some in simulation

- SISO BOMS, ... (and DEVS)

DEVS is hierarchical

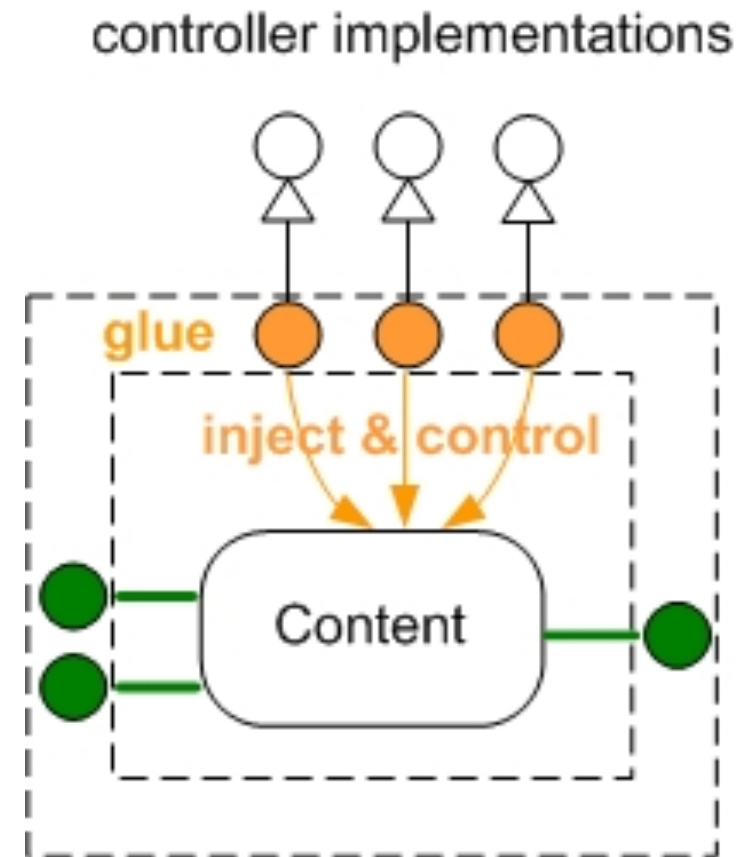
- Same as Fractal...

The Fractal Component Model

(E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, 2001)

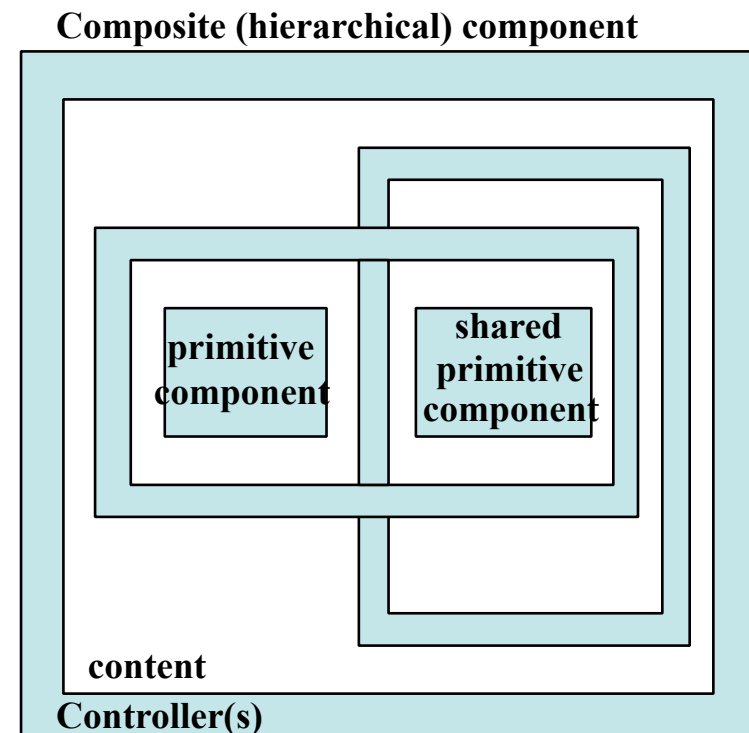
A Fractal Component

- A component is made of
 - A Content part
 - Business code
 - model implementation
 - A set of Controllers
 - Non functional part
- Controllers (*membrane*) may
 - intercept content's interactions
 - Reify method calls into events
 - provide services to inner part
 - Simulation API
 - provide services to outer part

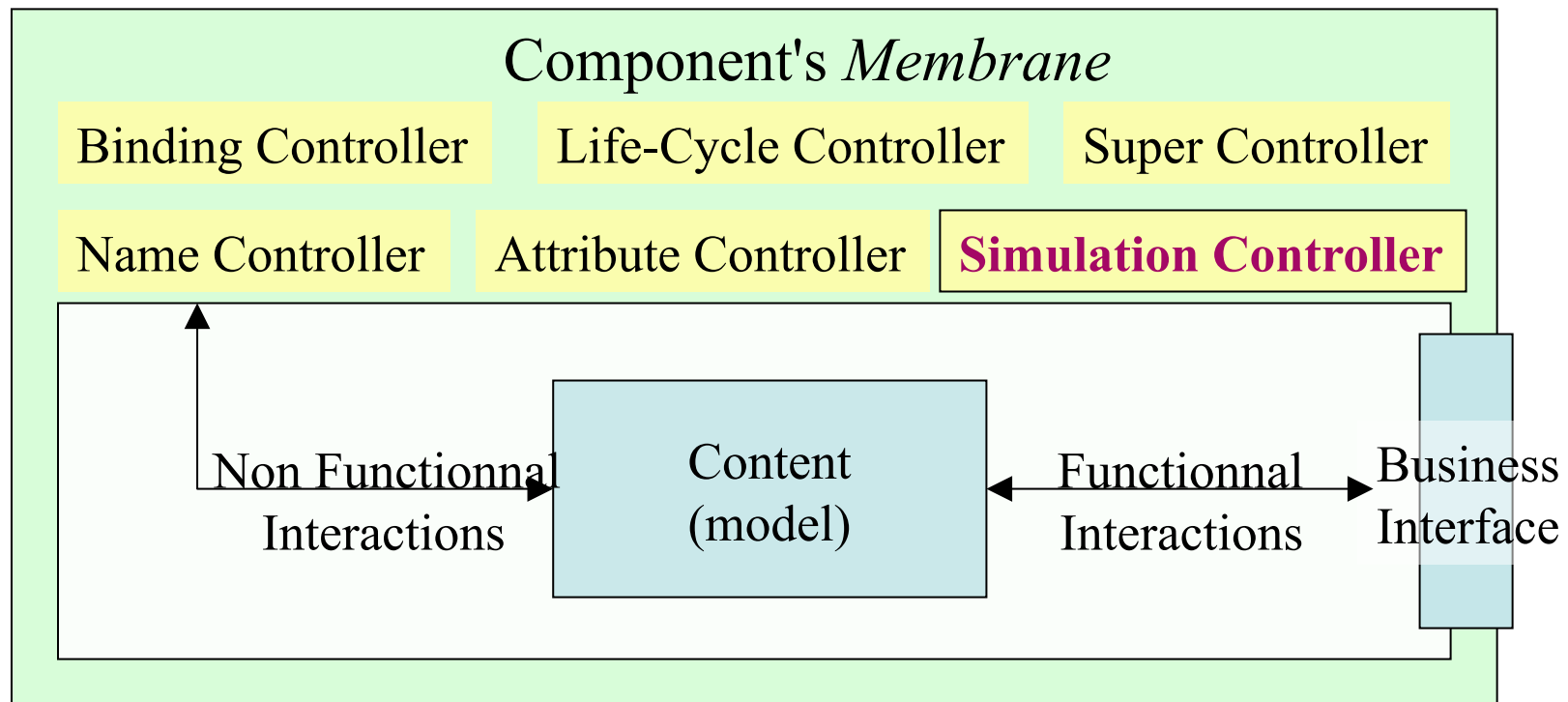


Fractal Architectural Concepts

- Primitive Component
 - Code Container
 - Client-server interactions
- Composite component
 - Hierarchical grouping
 - Strong Isolation
 - Shared sub-Component
- Dynamic (re)configuration
 - Factory Components
 - Dynamic bindings
- Introspection
- Extensibility of non-functional services
 - Controllers
- **Architecture Description Language**



Use Fractal in Simulation ?



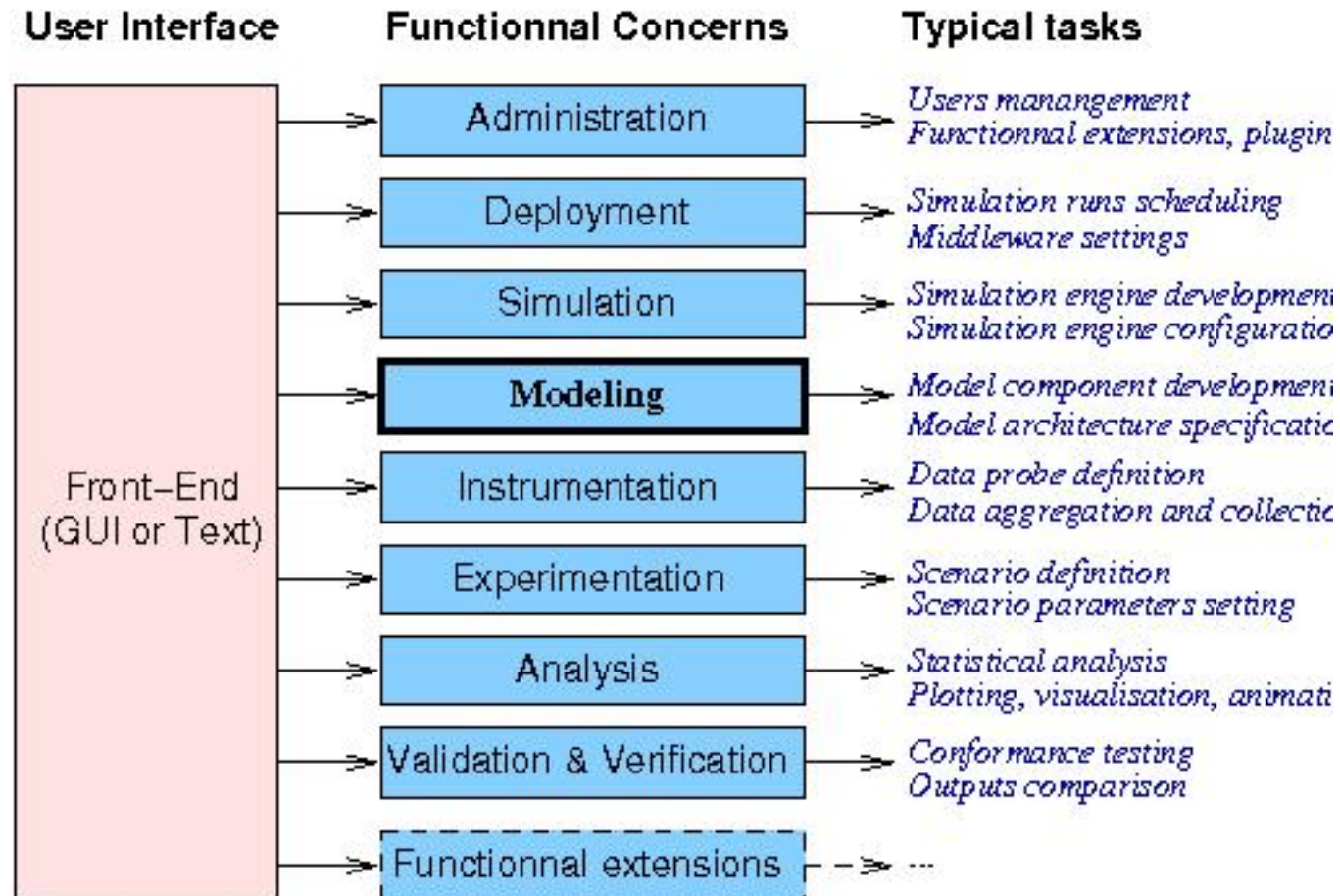
•Philosophy

- Reuse, reuse, reuse!
 - It already exists and it's possibly (much) better
 - **Layered Architecture**
- Be flexible (aka the Eclipse way)
 - being popular works better than being tyrannic
 - Let people do have what they need
 - offer choices rather than constraints

•OSA is a collection of (reused) tools

- AOKell, the Koch (membrane compiler)
- **FractalADL**
- Maven
- Scave, James,

OSA's Ambition: Cover All Aspects of a Simulation Study with Reused (Stolen) Stuff



OSA's Pillars

- Fractal Components

- Behave like regular objects (interface, method calls, ...)
- Hierarchical (with funny features)
- Malleable membranes

- Apache Maven


- Provides (hierarchical / multi-administrated) repository
- Dependency management
- Versionning
- Building capability
- Many more (actually a project management tool)

- FractalADL...**

- Aspect Oriented Programming

FractalADL

- OSA (Fractal)'s Architecture Description Language
- XML based
- OO-like Composition
 - Heritage
 - Overloading
- Extensible Language
 - Reflexive design
 - Factory parser is a
Fractal component assembly
 - Extended for simulation
 - Simulation controller
 - Component Instrumentation

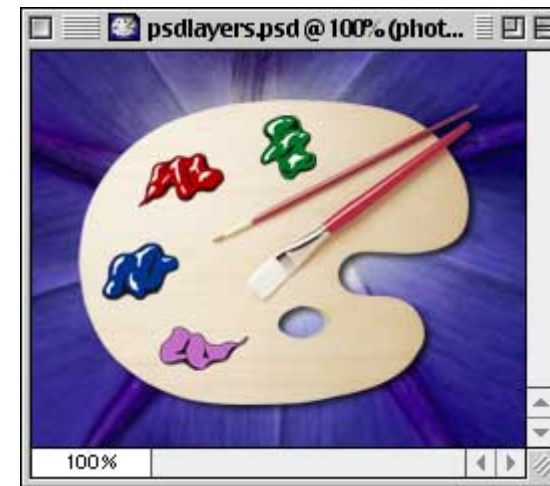
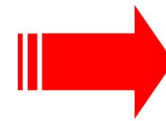
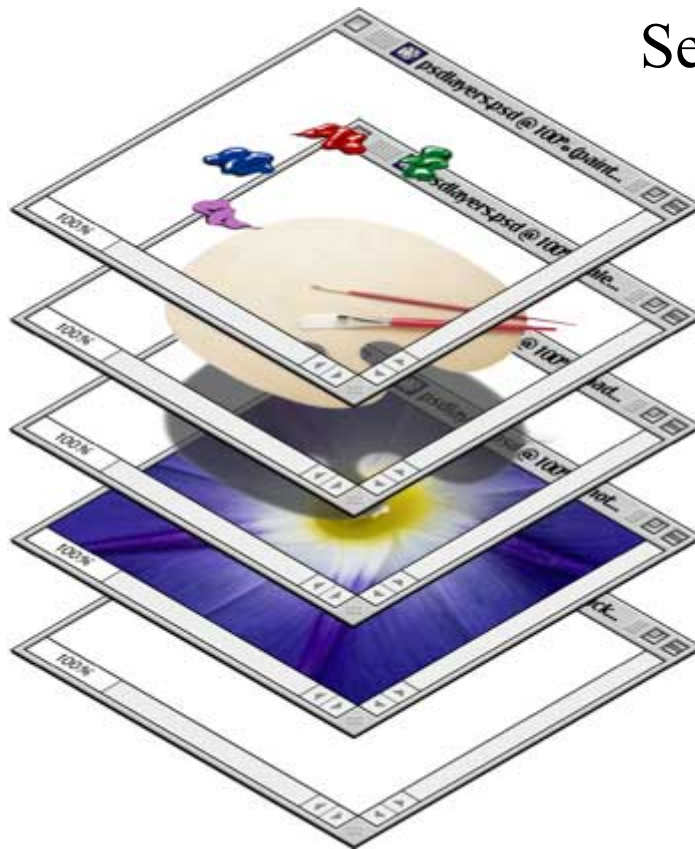


```

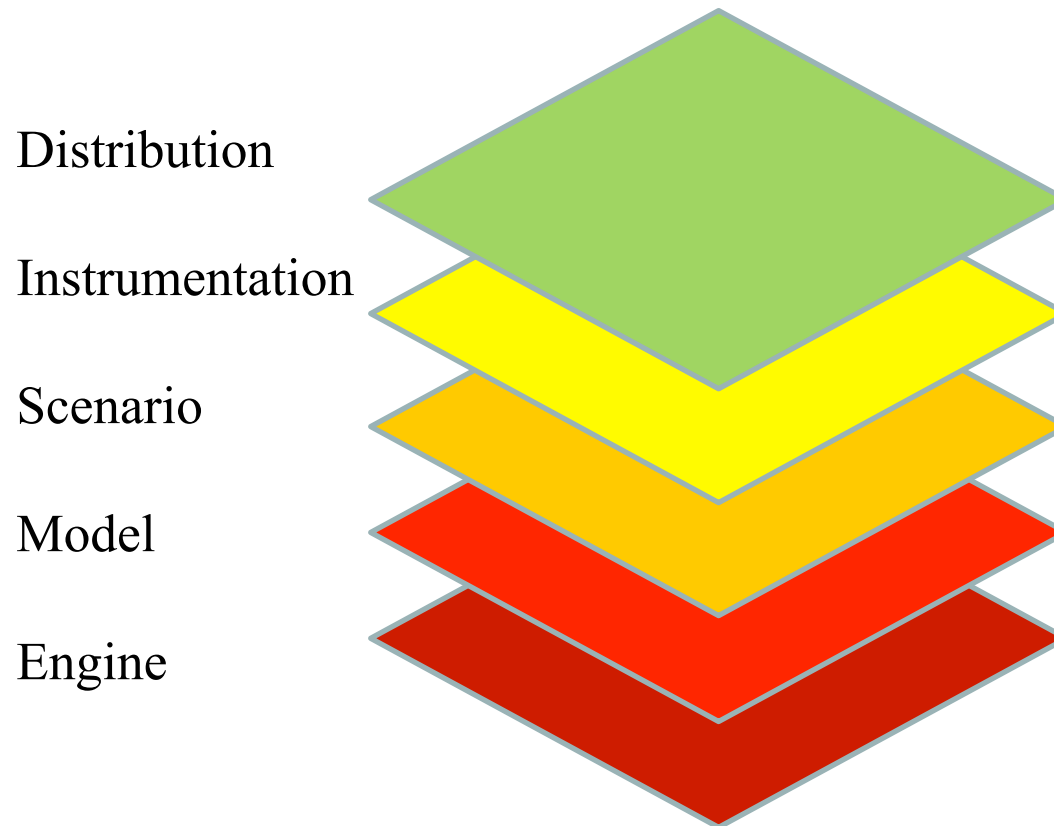
model-helloscale.fra  distributed-hello.fr  *model-helloworld.fr
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE definition PUBLIC "-//objectweb.org//DTD Fractal ADL 2.0"//EN>
<definition name="fr.inria.osa.helloworld.modelings.model-hello" >
  <component name="Hello"
    definition="fr.inria.osa.helloworld.modelings.Hello">
  </component>
  <component name="World"
    definition="fr.inria.osa.helloworld.modelings.World"/>
  <binding client="Hello.world" server="World.world" />
</definition>
    
```

Layered (De)Composition

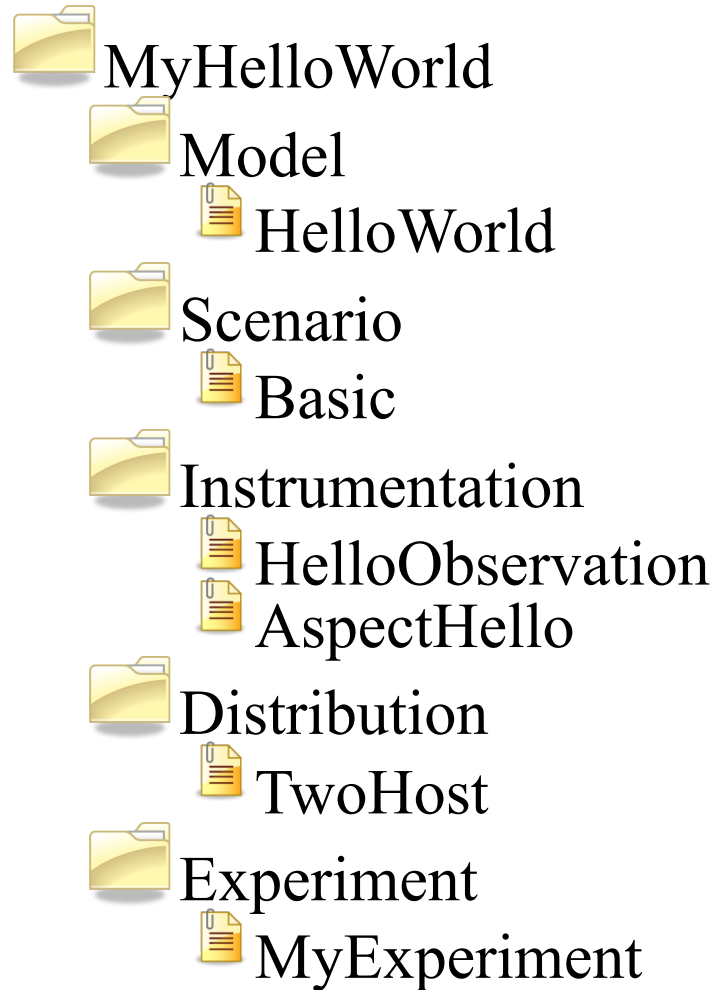
Separated concerns = more reuse



OSA's Separated/Reusable Concerns (so far)



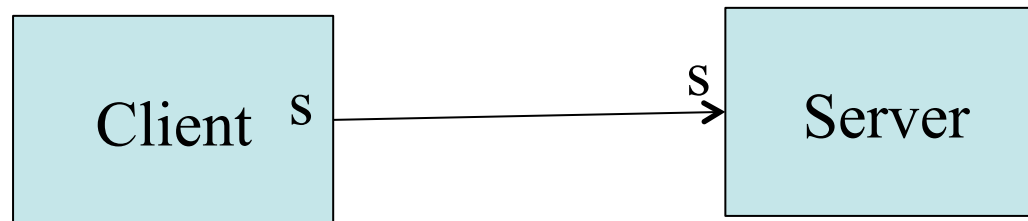
Layout: Each Concern is a Separate Maven (sub)Project



FractalADL (with additions for Simulation)

OSA/Fractal ADL Example

```
<definition name="cs.adl.HelloWorldModel">  
  <component name="client" definition="cs.adl.ClientImpl"/>  
  <component name="server" definition="cs.adl.ServerImpl" />  
  <binding client="client.s" server="server.s" />  
</definition>
```



ADL Read From Multiple Files

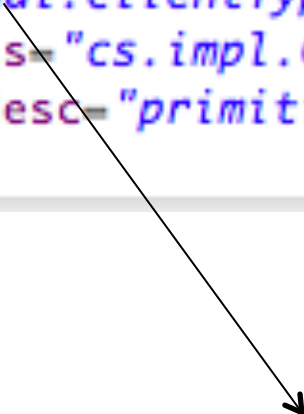
```
<definition name="cs.adl.HelloWorldModel">
  <component name="client" definition="cs.adl.ClientImpl"/>
  <component name="server" definition="cs.adl.ServerImpl" />
  <binding client="client.s" server="server.s" />
</definition>
```

```
<definition name="cs.adl.ClientImpl"
  extends="cs.adl.ClientType">
  <content class="cs.impl.ClientImpl" />
  <controller desc="primitiveSim" />
</definition>
```

```
<definition name="cs.adl.ServerImpl" extends="cs.adl.ServerType">
  <content class="cs.impl.ServerImpl" />
  <attributes signature="cs.impl.ServiceAttributes">
    <attribute name="header" value="-&gt;" />
    <attribute name="count" value="1" />
  </attributes>
  <controller desc="parametricPrimitiveSim" />
</definition>
```

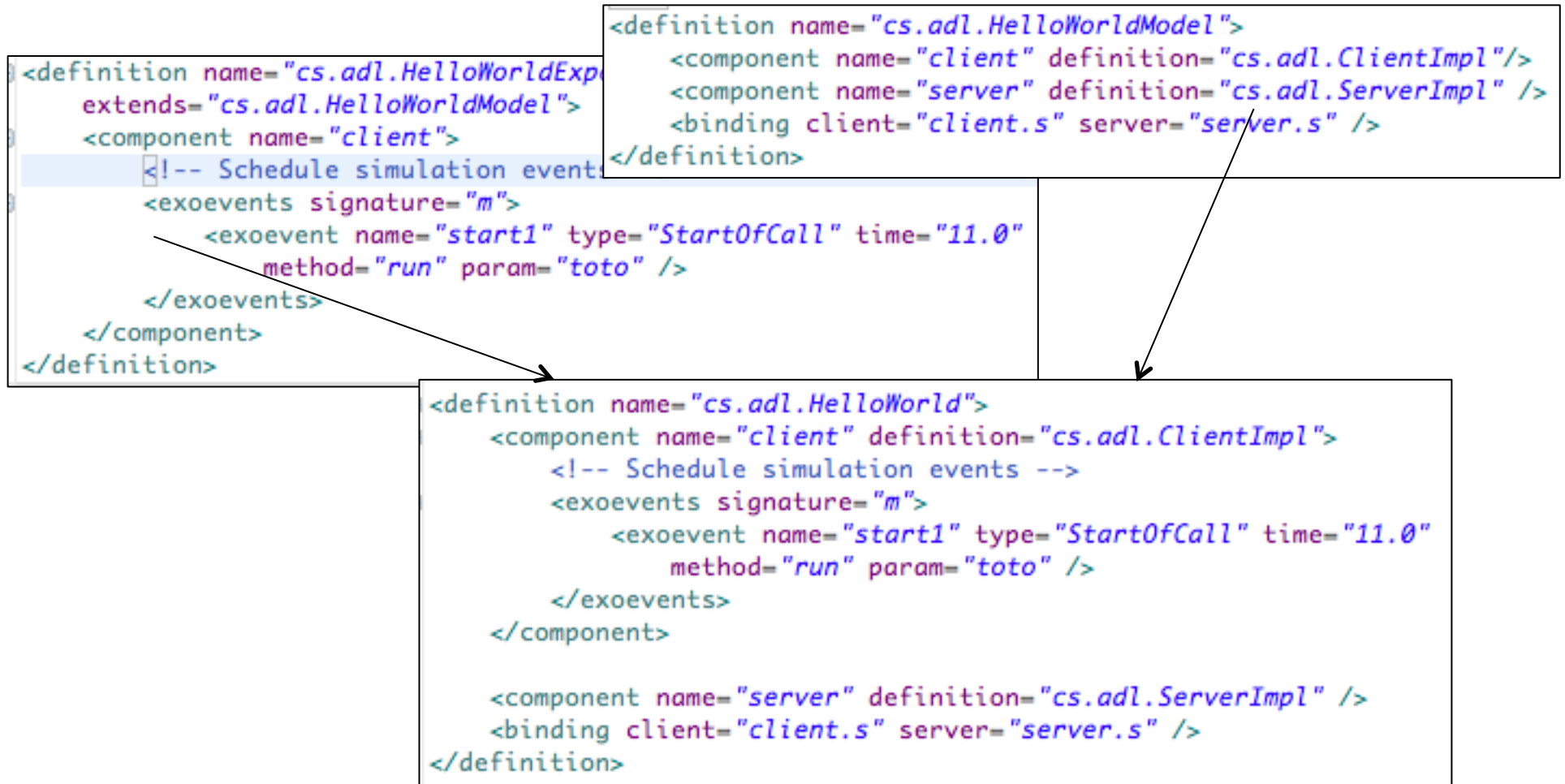
ADL Supports Heritage...

```
<definition name="cs.adl.ClientImpl"  
  extends="cs.adl.ClientType">  
  <content class="cs.impl.ClientImpl" />  
  <controller desc="primitiveSim" />  
</definition>
```

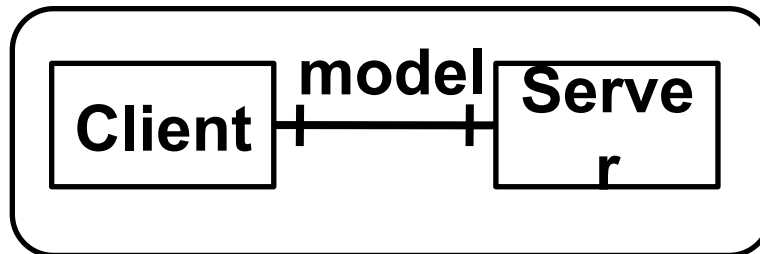


```
<definition name="cs.adl.ClientType">  
  <interface name="s" role="client" signature="cs.impl.Service" />  
  <interface name="m" role="server" signature="cs.impl.Starter" />  
</definition>
```

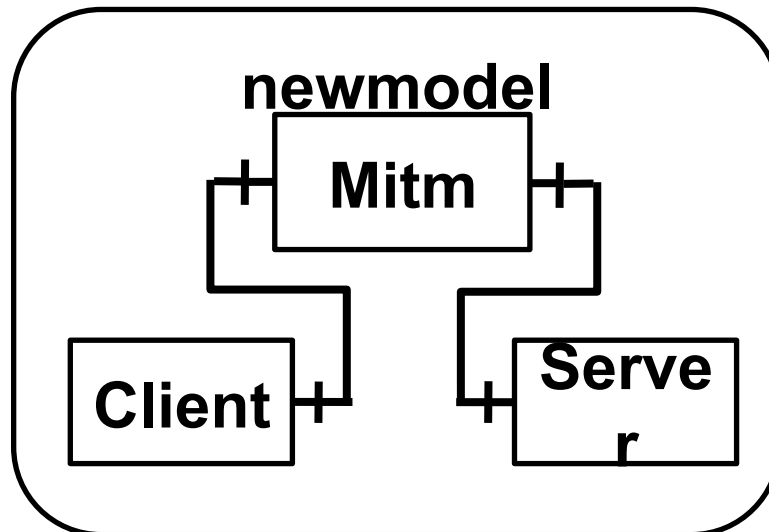
... and Overloading



Example of Advance Use: Man in the Middle Scenario

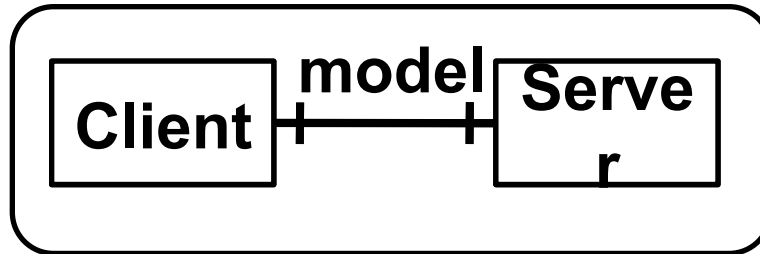


Definition name=**model**
 component name=**Client**
 component name=**Server**
 binding **Client -> Server**

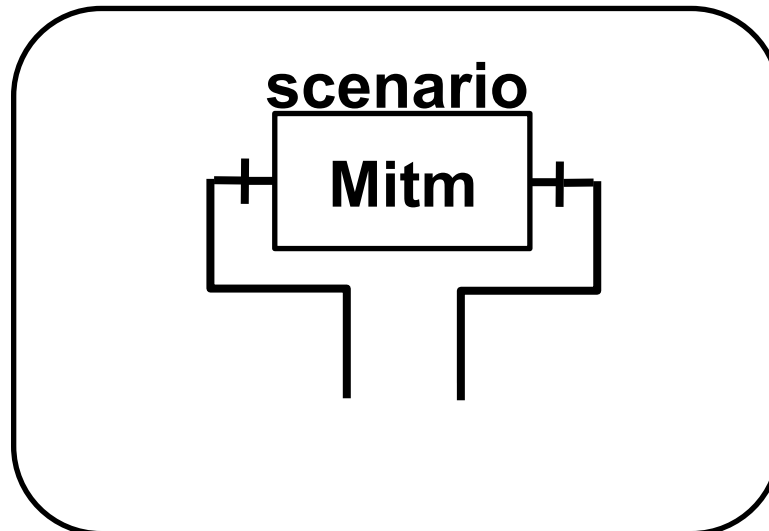


Definition name=**newmodel**
 component name=**Mitm**
 component name=**Client**
 component name=**Server**
 binding **Client -> Mitm**
 binding **Mitm -> Server**

Man in the Middle (With Heritage)

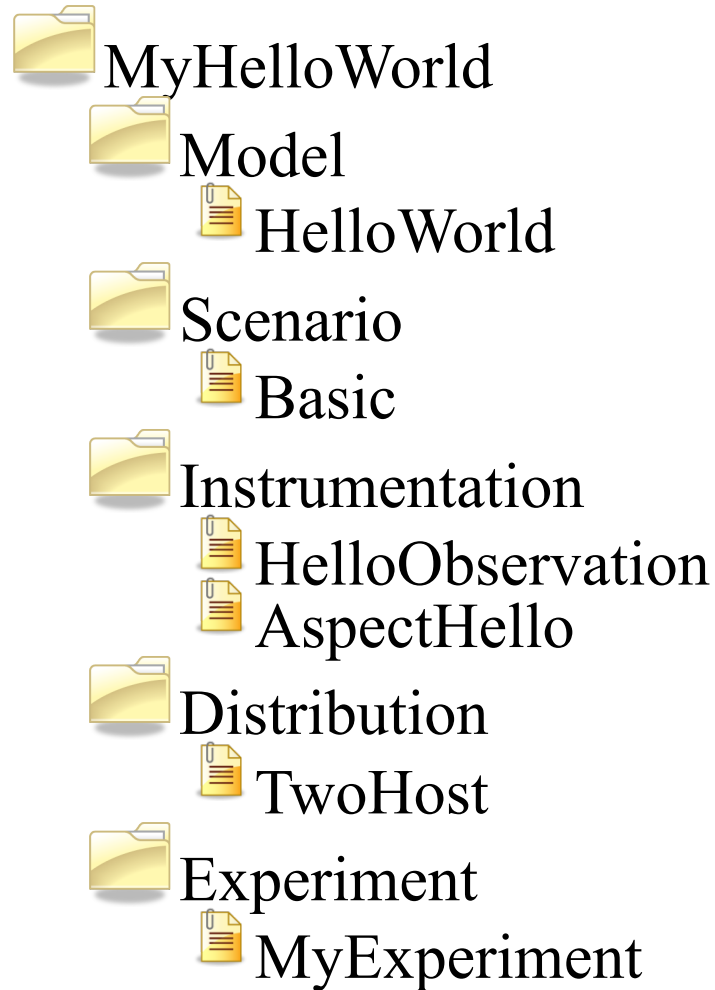


Definition name=**model**
component name=**Client**
component name=**Server**
binding **Client -> Server**



Definition name=**scenario**
extends **model**
component name=**Mitm**
binding **Client -> Mitm**
binding **Mitm -> Server**

Layout: Each Concern is a Composed using FractalADL



Conclusion - Claims

OSA/FractalADL shows that:

- Lots of things around already
 - Claim: EVERYTHING there already!
 - Repositories, versioning, reuse enablers, ...
 - Comp. Scientist: we MUST find it (software-bibliography)
- We can extend the scope of discussion to more than just Model+EF
 - Claim: we SHOULD extend the scope
- We can apply this to DEVS
 - Proof-of-concept: Reusing Rostock DEVS Engine (and much more)

Conclusion – Perspectives (1)

Add Workflows to OSA Architecture

- Improve Architecture for Better Integration of Sim. LifeCycle
- Work starting on Simulation Workflows
 - Express workflows
 - Using BPMN / BOS
 - Offers connectors to any real world task
 - Track steps followed by a study
 - Perspective: saves from over-documenting?
 - Support workflows
 - Re-run workflow
 - after bug identification
 - reproducibility
 - Extend workflow (new study)

Conclusion – Perspectives (2)

Steal-Reuse more and more

- Tools (eg. AKAROA)
- Engines
- Visualization (already Omnet++/Scave)
- Models
 - Work starting on interfacing NS3 engine
- Eclipse Support
 - Visualization/editing of zillions of XML files...

ADL Editor Plugin for Eclipse

