Some Desired Features for the DEVS Architecture Description Language

Olivier Dalle – Judicael Ribault

DEVS/TMS, Boston, April 6th 2011
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?
A simulation study workflow

Some philosophical questions:
• What is part of DEVS and what is not?
  • Conceptual Modeling
  • Modeling (implemented)
  • Simulation
  • Experimentation (partly)
• What is a standard used for?
  • Interoperability/reuse
• Can a global standard emerge from standardized sub-elements?
  • Hopefully...
Standardization of Models

Conceptual level

Atomic DEVS =

\[ < S, X, Y, \delta_{int}, \delta_{ext}, \lambda, ta > \]

Example:

\[ GEN = < S, X, Y, \delta_{int}, \delta_{ext}, \lambda, ta > \]

\begin{itemize}
  \item \( X = \phi \)
  \item \( Y = \{\text{out}\} \)
  \item \( S = \{G\} \)
  \item \( \delta_{int}: \delta_{int}(G) = G \)
  \item \( \delta_{ext}: \text{unavailable} \)
  \item \( \lambda(G) = \text{out} \)
  \item \( ta(G) = \text{GEN\_TIME} \)
\end{itemize}

Implementation Level ??

Atomic DEVS = a derived class?
which language?

Example (DEVJava):

```java
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
Layered (De)Composition

Separated concerns = more reuse
OSA's Separated/Reusable Concerns (so far)

Distribution
Instrumentation
Scenario
Model
Engine
Layout: Each Concern is a Separate Maven (sub)Project

- MyHelloWorld
  - Model
    - HelloWorld
  - Scenario
    - Basic
  - Instrumentation
    - HelloObservation
    - AspectHello
  - Distribution
    - TwoHost
  - Experiment
    - MyExperiment
FractalADL (with additions for Simulation)
OSA/Fractal ADL Example

```xml
<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...

```xml
<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

```xml
<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>
```

```xml
<definition name="cs.adl.HelloWorld">
  <component name="client" definition="cs.adl.ClientImpl">
    <!-- Schedule simulation events -->
    <exoevents signature="m">
      <exoevent name="start1" type="StartOfCall" time="11.0"
                method="run" param="toto"/>
    </exoevents>
  </component>
</definition>
```
Example of Advance Use: Man in the Middle Scenario

Definition name=model
component name=Client
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=\textit{model}
component name=Client
binding Client -> Server

Definition name=\textit{scenario}
extends model
component name=Mitm
binding Client -> Mitm
binding Mitm -> Server
Layout: Each Concern is a Composed using FractalADL

- MyHelloWorld
  - Model
    - HelloWorld
  - Scenario
    - Basic
  - Instrumentation
    - HelloObservation
    - AspectHello
  - Distribution
    - TwoHost
  - Experiment
    - MyExperiment
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 exend 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
• Vizualisation (already Omnet++/Scave)
• Models
  • Work starting on interfacing NS3 engine
• Eclipse Support
  • Vizualization/editing of zillions of XML files...
ADL Editor Plugin for Eclipse