Binding Layers Level 0
An abstract multi-purpose component layer

Olivier Dalle

Université Nice Sophia Antipolis
I3S UMR CNRS 7172& INRIA CRI-SAM
2004 Route des Lucioles - BP 93
06903 Sophia Antipolis, FRANCE

olivier.dalle@unice.fr

SCADA Workshop
Sophia Antipolis, November 28, 2013
Introduction

- Previous work: OSA Project (with J. Ribault)
- Simulation-oriented components: DEVS

Problem Statement

- Level 0: an Abstract Component Model
- Other Issues addressed in Binding Layers Project

Binding Layers Explained

- Level 0
- Upper Levels

Conclusions & Perspectives
Motivations

“Build from scratch or reuse?”

- There is no perfect simulator BUT
  - All the elements of your perfect simulator probably exist.
  - if not, build only the missing part!

“Can we trust our simulation results?”

- Trust comes from validation step (cf. VV&A)
- More reusing $\rightarrow$ less validation

“Which credibility in comparing results with others studies?”

- More sharing $\rightarrow$ more credibility
Objectives

- Separation of modeling concerns
  - $\rightarrow$ component-based framework
- Separation of simulation concerns
  - $\rightarrow$ layered approach
- Bridge between concerns
  - $\rightarrow$ aspect-oriented programming
- Backup and replayability
  - $\rightarrow$ maven project management

**GOAL**

- build from or reuse existing parts from others simulators and third-party tools
Open Simulation Architecture
A component-based framework

OSA extends Fractal Components
Open Simulation Architecture
A layered approach
Open Simulation Architecture
aspect-oriented programming

OOP

OOP + AOP

Model source
Instrumentation X
Instrumentation Y
Instrumentation X

Model source
Instrumentation X

Aspect source
Instrumentation X

Aspect source
Instrumentation Y

AOP
WEAVER

Functional Code
Code to instrument variable X
Code to instrument variable Y

Olivier Dalle
Binding Layers Level 0
DEVS formalism
Anatomy of a component

Components are similar to objects

- Ports/Connectors: Data-flow (carry bags of events)
- Strong semantic of operation:
  - $X, Y, S$: Input events, Output events, States
  - $\delta_{int}: S \rightarrow S$ (Internal transition function)
  - $\delta_{ext}: S \times X \times T \rightarrow S$ (External transition function)
  - $\lambda: S \rightarrow Y$ (Output function)
  - $ta: S \rightarrow T$ (Time advance function)
DEVS formalism

DEVS Assemblies

- Strong Semantics
  \[< X, Y, D, \{ M_i \}, \{ I_i \}, \{ Z_{ij} \}, sel >\]
  - X, Y: Inputs and outputs
  - D: Set of names
  - \( M_i, i \in D \): basic DEVS

- Hierarchical: atomic vs. coupled
- Oriented Bindings
- Multi-points Bindings
- Static binding
Very Large component-based architectures

Large architectures are quite common in simulation...

- Global world-size telecommunications
- Road traffic
- Body cells
- Particles
- ...

...New web-based services reach unprecedented scale

- Today: Twitter, Google, FB, ...
- Tomorrow??
How to build very large component-based architectures?

North face approach...
- Choose a component model
- Read docs...
  ...scratch your head for long hours...
- ...Build hello world example
  (without following tutorial!)
- Design the big architecture...
  ...Using proprietary tools/language
  ...and pray.

What if result it is not satisfactory?
RESTART from scratch?
- Component content (code) might be reusable...
- ...but the architecture?
Separation of Concerns

Architecture vs. Behavior
- Architecture = assemblies
- Behavior = code

Simulation concerns
- Model subject of study
- Build scenario(s)
- Instrument, Observe
- Collect and process data
- Distribute execution
- ...

Software (engineering) concerns
- Security, confidentiality
- Persistence
- Reconfiguration/life cycle
- Fault detection/recovery
- Real time
- Debugging
- ...

How to Separate Concerns?
And REUSE them!
Example of a many concerns simulation

Simulation of a “Digital City” project

- Water Flooding
- Fire Spreading
- Electrical Network
- Water Network
- Road Network
- Data Network

- System Model
- Scenario model
- Observation
- On-line processing
- Causal Model
- Validation
- Distributed execution
- Data-mining
- Visualization

Modeling Layers
Simulation Layers
Observation Layer
Road Network Layer
The problem with hierarchy

A choice has to be made
What is outside? (implies what is inside)
How to focus only on the "inside" level?
- Deeply buried
- Scattered ...
**Unified Architecture API**

**My Dream Component API**

1. `model=InitModel("AModel.cfg", "path/to/app.cfg")`
2. `hello=model.CreateInstance("Hello")`
3. `world=model.CreateInstance("World")`
4. `model.Connect(hello,world)`
5. `model.Go!()`

**Challenge**

Make it work with *(m)any* component model!
Core Principles of API

Main Assumptions/Constraints

- Components have object semantics
- Bindings can be multi-points
- Bindings are oriented
- Hide any specific details
- Construction support mandatory
- Destruction support optional
Specific Issues to be Address

- Sequence of operations
- Attachment details
- Details of instance creation
- Unsupported operations
Binding Layers Level 0 Specification

architecture specification

requests sent to LCM

Level 0 API

EBS  RTE  CIS

ISS

resolution table(s)

CSS  OSS

(LCM)

(LCM)

component

binding

realization

Demo!

(previous version...)

Olivier Dalle
Level 1
Dynamicity support

Add support for programming layers

- Built on top of Level 0 API
- Provide a simple scripting language
- Support for automatic configuration

Instructions

- Bootstrap(type, count)
- Induce(predicate, type, count)
- Bind(Predicate, Predicate)

Layer Definition:

- Bootstrap(white)
- Induce('C<2', white, count=3)
- Induce('C>0', white, count=3)
- Induce('C≥2 ∧ white', black, count=2)
Level 1

Example

Layer Instructions:

- Bootstrap(C)
- Bootstrap(C)
- Induction('=C_1,C')
- Binding('=C_1,2','=C_3')

Layer Instructions:

- Bootstrap(C)
- Bootstrap(C)
On-demand instance creation

Induce_or_connect

(1) induce_or_connect('client.*','server','client:server')
(2) induce('server.*','worker','server:worker',count=5)
(3) induce_or_connect('worker.*','DB','worker:db')
Add support for multiple layers

- A layer can extend another
- Extender can replace/remove extendee’s instructions
- Extender can share with extendee

Special feature

NO CLOSURE: a layer is NOT a component → NO Hierarchy
CLAIM: Hierarchy is believed to be hindering Separation of Concern.
SUBSTITUTE: Component Sharing
Level 2
Example of sharing & extension

Layer A, extends B, C

Layer B

Layer C

Layer A, extends B, C

Layer B

Layer C

Olivier Dalle
Level 3
Layer Algebra

Origin layer definition:
Layer "Foo", dispatch=fc(C), count=4
  Bootstrap(...)
  Induce(...)
  Bind(...)

Top Layer definition:
Layer "Foo-top", extends "Foo-{1−4}"
  Bootstrap(...)
  Induce(...)
  Bind(...)

Sub-Layer definitions:
Layer "Foo−1":
  Bootstrap(...)
  Induce(...)
  Bind(...)

Layer "Foo−2":
  Bootstrap(...)
  Induce(...)
  Bind(...)
Conclusions

Benefits of Level0 Abstract API

- Allows switching model
- Testing/debugging
- Simple programming
- Self-contained
- Base for bigger things
Almost there!

- Specification complete (Draft soon available)
- Some early implementation (Java)
  - Fractal
  - Dummy
- More validation to come
  - DEVS
  - Process-and-pipe
Work in progress...

- Level 1: working on specs and proto
- Level 2: seems to work on paper
- Level 3: some ideas need further thinking
- Level 4: maybe an aspect language for weaving layers?
Thank you!

Questions and comment are welcome!
Components are POSIX process
Component have a (system) context
Ports/Connectors: file descriptors
Content: “business” code
API: System calls
POSIX Process-and-pipes
Component assemblies

- Flat structure
- Oriented Bindings
- Multi-points Bindings

- Dynamic Bindings
  - (re-)configuration tricky
    - opening sequence
    - deadlocks...