Some Methodology Issues

And Methodology Experiments in the OSA Project

Olivier DALLE[1] (parts stolen to Judicael Ribault[1])

[1] INRIA - CRISAM, University of Nice Sophia Antipolis
I3S-UMR CNRS 6070, BP93 - 06903 Sophia Antipolis, France
judicael.ribault@sophia.inria.fr - olivier.dalle@sophia.inria.fr

Journees SUD, Cargese, April 2010

Agenda

A selected number of issues and possible solutions:

- How to code discrete time ?
- How to Ensure Reproduce-ability, Trace-ability and Durability?
- How to Instrument a Simulation?
  - Our solution: The OSIF Framework

Agenda

A selected number of issues and possible solutions:

- How to code discrete time ?
- How to Ensure Reproduce-ability, Trace-ability and Durability?
- How to Instrument a Simulation?
  - Our solution: The OSIF Framework

Agenda

A selected number of issues and possible solutions:

- How to code discrete time ?
- How to Ensure Reproduce-ability, Trace-ability and Durability?
- How to Instrument a Simulation?
  - Our solution: The OSIF Framework

Agenda

A selected number of issues and possible solutions:

- How to code discrete time ?
- How to Ensure Reproduce-ability, Trace-ability and Durability?
- How to Instrument a Simulation?
  - Our solution: The OSIF Framework

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?

Conclusion: Floating point seems a good idea. . .

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit ? nano-second?
    Yes, at least: CPU run at GHz frequency, Network transmit TBits/s,
    nanosec = 10^{-9} sec, hence 2^{32} is not enough for more than a few seconds.

Conclusion: Floating point seems a good idea. . .

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, . . .
  - nanosec = 10^{-9} sec, hence 2^{32} is not enough for more than a few seconds

Conclusion: Floating point seems a good idea. . .

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, ...
  - nanosec = 10^{-9} sec, hence 2^{32} is not enough for more than a few seconds.

Conclusion: Floating point seems a good idea. . .

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, ...
  - nanosec $= 10^{-9}$ sec, hence 2^{32} is not enough for more than a few seconds.
    - At least 64 bits needed...
      seems more expensive to use integers than Floats

Conclusion: Floating point seems a good idea...

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, ...
  - nanosec = 10^{-9} sec, hence 2^{32} is not enough for more than a few seconds.
    - At least 64 bits needed...
    - seems more expensive to use integers than Floats

Conclusion: Floating point seems a good idea...

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, . . .
  - nanosec $= 10^{-9}$ sec, hence 2^{32} is not enough for more than a few seconds.
    - At least 64 bits needed. . .
    - seems more expensive to use integers than Floats

Conclusion: Floating point seems a good idea. . .

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, ...
  - nanosec $= 10^{-9}$ sec, hence 2^{32} is not enough for more than a few seconds.
    - At least 64 bits needed...
    - seems more expensive to use integers than Floats

Conclusion: Floating point seems a good idea...

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, ...
  - nanosec $= 10^{-9}$ sec, hence 2^{32} is not enough for more than a few seconds.
    - At least 64 bits needed...
    - seems more expensive to use integers than Floats

Conclusion: Floating point seems a good idea...

How to Code Discrete Time?

What Coding Options Do We Have?

- Floating Point Numbers?
  - 32 (1+8+23) bits ? 64 (1+11+52) bits ? 128 (1+15+112) bits?
  - Accurate?
  - SURE! 32 bits seems accurate enough for ANY duration.
- Integers ?
  - 32 bits ? Which unit? nano-second?
  - Yes, at least, CPU run at GHz Frequency, Network transmit TBits/s, ...
  - nanosec $= 10^{-9}$ sec, hence 2^{32} is not enough for more than a few seconds.
    - At least 64 bits needed...
    - seems more expensive to use integers than Floats

Conclusion: Floating point seems a good idea...

How to Code Discrete Time?

Let's experiment a bit with FP numbers.

- Assume time unit is second.
- Assume after 1 hour (virtual/sim time)
  - one event is scheduled after 50 nanosec
  - other events start to be scheduled every nanosec
- (eg. python)

- Ok, ok, funny little glitch. But, is floating point inadequate?

How to Code Discrete Time?

Let's experiment a bit with FP numbers.

- Assume time unit is second.
- Assume after 1 hour (virtual/sim time)
  - one event is scheduled after 50 nanosec
  - other events start to be scheduled every nanosec
- (eg. python)

```
y=3600+50e-9
z=3600.0
for i in xrange(50):  z=z+1.0e-9
```

  $y \rightarrow 3600.000000049999$
  $z \rightarrow 3600.000000049995$

- Ok, ok, funny little glitch. But, is floating point inadequate?

How to Code Discrete Time?

Let's experiment a bit with FP numbers.

- Assume time unit is second.
- Assume after 1 hour (virtual/sim time)
  - one event is scheduled after 50 nanosec
  - other events start to be scheduled every nanosec
- (eg. python)

```
y=3600+50e-9
z=3600.0
for i in xrange(50):  z=z+1.0e-9
```

$y \rightarrow 3600.000000499999$

$z \rightarrow 3600.000000499995$

- Ok, ok, funny little glitch. But, is floating point inadequate?

How to Code Discrete Time?

Let's experiment a bit with FP numbers.

- Assume time unit is second.
- Assume after 1 hour (virtual/sim time)
  - one event is scheduled after 50 nanosec
  - other events start to be scheduled every nanosec
- (eg. python)

```
y=3600+50e-9
z=3600.0
for i in xrange(50):  z=z+1.0e-9
```

$y \rightarrow 3600.0000000499999$

$z \rightarrow 3600.0000000499995$

- Ok, ok, funny little glitch. But, is floating point inadequate?

How to Code Discrete Time?

Let's experiment a bit with FP numbers.

- Assume time unit is second.
- Assume after 1 hour (virtual/sim time)
  - one event is scheduled after 50 nanosec
  - other events start to be scheduled every nanosec
- (eg. python)

```
y=3600+50e-9
z=3600.0
for i in xrange(50):  z=z+1.0e-9
```

y → 3600.0000000499999

z → 3600.0000000499995

- Ok, ok, funny little glitch. But, is floating point inadequate?

How to Code Discrete Time?

Let's experiment a bit with FP numbers.

- Assume time unit is second.
- Assume after 1 hour (virtual/sim time)
  - one event is scheduled after 50 nanosec
  - other events start to be scheduled every nanosec
- (eg. python)

```
y=3600+50e-9
z=3600.0
for i in xrange(50):  z=z+1.0e-9
```

  y → 3600.0000000499999
  z → 3600.0000000499995
- Ok, ok, funny little glitch. But, is floating point inadequate?

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine. . .
- Order Sequence of Events
  - E_1 happens at $t = T1$; E_2 happens at $t = T2$, $T2 > T1$

Is accuracy of Time/Dates Important Here?
YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ. . .

Violates the Sequential Order
Eg. NS-2 claims tie events are processed FIFO
This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events
  - E_1 happens at $t = T1$, E_2 happens at $t = T2$, $T2 > T1$
  - Is accuracy of Time/Dates Important Here?
  - YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ ...
    - Violates the Sequential Order
    - Eg: NS-2 claims tie events are processed FIFO
    - This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events
  - E_1 happens at $t = T1$, E_2 happens at $t = T2$, $T2 > T1$
    - Hence, obviously (!), E_1 happens before E_2.
  - Is accuracy of Time/Dates Important Here?
  - YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ ...
    - Violates the Sequential Order
    - Eg. NS-2 claims tie events are processed FIFO
    - This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events
  - E_1 happens at $t = T1$, E_2 happens at $t = T2$, $T2 > T1$
    - Hence, obviously (!), E_1 happens before E_2.
  - Is accuracy of Time/Dates Important Here?
  - YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ ...
    - Violates the Sequential Order
    - Eg. NS-2 claims tie events are processed FIFO
    - This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events
  - E_1 happens at $t = T1$, E_2 happens at $t = T2$, $T2 > T1$
    - Hence, obviously (!), E_1 happens before E_2.
  - Is accuracy of Time/Dates Important Here?
  - YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ ...
    - Violates the Sequential Order
    - Eg. NS-2 claims tie events are processed FIFO
    - This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events
  - E_1 happens at $t = T1$, E_2 happens at $t = T2$, $T2 > T1$
    - Hence, obviously (!), E_1 happens before E_2.
  - Is accuracy of Time/Dates Important Here?
  - YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ ...
    - Violates the Sequential Order
    - Eg. NS-2 claims tie events are processed FIFO
    - This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events
  - E_1 happens at $t = T1$, E_2 happens at $t = T2$, $T2 > T1$
    - Hence, obviously (!), E_1 happens before E_2.
  - Is accuracy of Time/Dates Important Here?
  - YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ ...
    - Violates the Sequential Order
    - Eg. NS-2 claims tie events are processed FIFO
    - This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Code Discrete Time?

What is Time Used for in Simulations?

- Numerical Evaluation of Duration
  - $duration = end - start$
  - Is Accuracy of Time/Dates Important Here?
  - Not so much: duration +/- 0.0000005% is probably fine...
- Order Sequence of Events
  - E_1 happens at $t = T1$, E_2 happens at $t = T2$, $T2 > T1$
    - Hence, obviously (!), E_1 happens before E_2.
  - Is accuracy of Time/Dates Important Here?
  - YES! T1 +/- ϵ MAY be larger than T2 +/- ϵ ...
    - Violates the Sequential Order
    - Eg. NS-2 claims tie events are processed FIFO
    - This is a WRONG statement

Conclusion: I strongly recommend coding time using Int64 !

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
- When a bug is found in the code of an experiement

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
    - Even after a long time
- Colleagues are able to reproduce experiments

- When a bug is found in the code of an experiement

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement
  - We are able to identify which results are impacted
  - We know which published papers must be updated/invalidated

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement
  - We are able to identify which results are impacted
  - We know which published papers must be updated/invalidated

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement
  - We are able to identify which results are impacted
  - We know which published papers must be updated/invalidated

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement
  - We are able to identify which results are impacted
  - We know which published papers must be updated/invalidated

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement
  - We are able to identify which results are impacted
  - We know which published papers must be updated/invalidated

Hopeless???

How to Ensure Reproduce-ability, Trace-ability and Durability?

In an ideal world:

- Results of experiments are published
- Publication was reviewed
- Reviewers were able to reproduce experiments
  - Even after a long time
- Colleagues are able to reproduce experiments
  - They are able to verify, detect flaws
  - They can work incrementally
- When a bug is found in the code of an experiement
  - We are able to identify which results are impacted
  - We know which published papers must be updated/invalidated

Hopeless???

How to Build an Ideal World?

Publishing a result is a **process**

- Start from an initial state (OS release, system config, . . .)
- (possibly) Build tools for experiment
- Experiment consist in multiple tasks, eg.:
  - Initial setup
  - Running experiment
  - Gathering data
  - Processing data
  - V,V & A
  - Generating plots, animation
  - Writing a paper
- Idea: What about formalizing experimental workflows?

How to Build an Ideal World?

Publishing a result is a **process**

- Start from an initial state (OS release, system config, . . .)
- (possibly) Build tools for experiment
- Experiment consist in multiple tasks, eg.:
  - Initial setup
  - Running experiment
  - Gathering data
  - Processing data
  - V,V & A
  - Generating plots, animation
  - Writing a paper
- Idea: What about formalizing experimental workflows?

How to Build an Ideal World?

Publishing a result is a **process**

- Start from an initial state (OS release, system config, ...)
- (possibly) Build tools for experiment
- Experiment consist in multiple tasks, eg.:
  - Initial setup
  - Running experiment
    - Follow experiment plan
    - Until some criteria is met...
  - Gathering data
  - Processing data
  - V,V & A
    - Possibly loop backard...
  - Generating plots, animation
  - Writing a paper
- Idea: What about formalizing experimental workflows?
  - See for example the myExperiment project

How to Build an Ideal World?

Publishing a result is a **process**

- Start from an initial state (OS release, system config, . . .)
- (possibly) Build tools for experiment
- Experiment consist in multiple tasks, eg.:
  - Initial setup
  - Running experiment
    - Follow experiment plan
    - Until some criteria is met. . .
  - Gathering data
  - Processing data
  - V,V & A
    - Possibly loop backard. . .
  - Generating plots, animation
  - Writing a paper
- Idea: What about formalizing experimental workflows?
  - See for example the *myExperiment* project. . .

How to Build an Ideal World?

Publishing a result is a **process**

- Start from an initial state (OS release, system config, . . .)
- (possibly) Build tools for experiment
- Experiment consist in multiple tasks, eg.:
  - Initial setup
  - Running experiment
    - Follow experiment plan
    - Until some criteria is met. . .
  - Gathering data
  - Processing data
  - V,V & A
    - Possibly loop backard. . .
  - Generating plots, animation
  - Writing a paper
- Idea: What about formalizing experimental workflows?
  - See for example the *myExperiment* project. . .

How to Build an Ideal World?

Publishing a result is a **process**

- Start from an initial state (OS release, system config, . . .)
- (possibly) Build tools for experiment
- Experiment consist in multiple tasks, eg.:
  - Initial setup
  - Running experiment
    - Follow experiment plan
    - Until some criteria is met. . .
  - Gathering data
  - Processing data
  - V,V & A
    - Possibly loop backard. . .
  - Generating plots, animation
  - Writing a paper
- Idea: What about formalizing experimental workflows?
  - See for example the *myExperiment* project. . .

How to Build an Ideal World?

Assuming workflows have been identified, what else do we need?

- Dependencies support management
- Archiving facilities
- Versionning, branch support
- Automation
- Most of these, we alreday have!

- But none of these is the panacea :-(
- We still have lot of work to do. . . :-)

How to Build an Ideal World?

Assuming workflows have been identified, what else do we need?

- Dependencies support management
- Archiving facilities
- Versionning, branch support
- Automation
- Most of these, we alreday have!
  - Makefiles, Maven, Ant, RPM, forges, repositories,
- But none of these is the panacea :-(
- We still have lot of work to do. . . :-)

How to Build an Ideal World?

Assuming workflows have been identified, what else do we need?

- Dependencies support management
- Archiving facilities
- Versionning, branch support
- Automation
- Most of these, we alreday have!
  - Makefiles, Maven, Ant, RPM, forges, repositories, ...
- But none of these is the panacea :-(
- We still have lot of work to do... :-)

How to Build an Ideal World?

Assuming workflows have been identified, what else do we need?

- Dependencies support management
- Archiving facilities
- Versionning, branch support
- Automation
- Most of these, we alreday have!
  - Makefiles, Maven, Ant, RPM, forges, repositories, . . .
- But none of these is the panacea :-(
- We still have lot of work to do. . . :-)

How to Build an Ideal World?

Assuming workflows have been identified, what else do we need?

- Dependencies support management
- Archiving facilities
- Versionning, branch support
- Automation
- Most of these, we alreday have!
  - Makefiles, Maven, Ant, RPM, forges, repositories, . . .
- But none of these is the panacea :-(
- We still have lot of work to do. . . :-)

How to Build an Ideal World?

Assuming workflows have been identified, what else do we need?

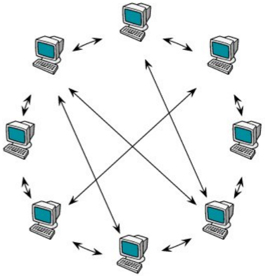
- Dependencies support management
- Archiving facilities
- Versionning, branch support
- Automation
- Most of these, we alreday have!
  - Makefiles, Maven, Ant, RPM, forges, repositories, . . .
- But none of these is the panacea :-(
- We still have lot of work to do. . . :-)

How to Build an Ideal World?

Assuming workflows have been identified, what else do we need?

- Dependencies support management
- Archiving facilities
- Versionning, branch support
- Automation
- Most of these, we alreday have!
  - Makefiles, Maven, Ant, RPM, forges, repositories, ...
- But none of these is the panacea :-(
- We still have lot of work to do. . . :-)

How to Instrument a Simulation?
The P2P use case



- Looking for parameters effects or validation ?
  - Edit source file, add instrumentation code, recompile
  - Run simulation/experiment
  - Grab all the observed/sampled data into one place
  - Post-process data

Common Instrumentation and Statistical Analyses Usage

The P2P use case

- Looking for parameters effects or validation ?
  - Edit source file, add instrumentation code, recompile
  - Run application
  - Grab all the instrumented data into one place
  - Post-process data
- Issues raised:
  - instrumentation and modeling concerns are mixed together
    - hard to maintain
    - issue when mixing with previous/other instrumentation
  - breaks model validation
    - loose some of the reuse benefits
  - bandwidth and memory overhead

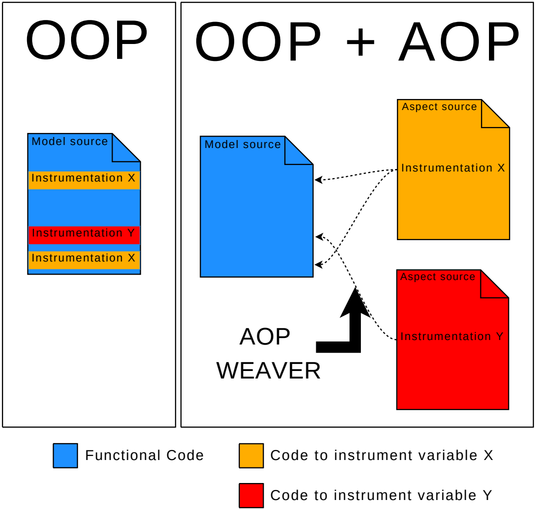
- Separate instrumentation concern from modeling concern
  $\Longrightarrow$ Aspect-Oriented Programming
- Process data during the simulation runs
  $\Longrightarrow$ COSMOS
- Reusable data processing
  $\Longrightarrow$ COSMOS is component-based
- Compose complex instrumentation and data processing
  $\Longrightarrow$ Architecture Description Language with multiple inheritance and overloading capability
- Validation results
  $\Longrightarrow$ use the same data processing both in simulations and experimentations

Separation of Concerns
(Using Aspect-Oriented Programming)

- Paradigm for modularizing applications with many concerns
- Aspect Oriented Programming
  - Instructions are placed in separate source files
- Identify particular instructions in an existing code (pointcut)
  - To apply pre/post/replacement processings
  - To enrich/extend existing code
- Apply aspect at compile-time or at run-time
- AOP exists for most programming languages:
  - C / C++, Java, C#, Perl, PHP, Python, Ruby, etc.
- Instrument your model and send values to COSMOS

Separation of Concerns

Aspect-oriented programming



Separation of Concerns

Aspect-oriented programming

- AspectJ example:

```
after (com.example.Peer peer) :
execution (void com.example.Peer.boot()) && this (peer)
{
  System.out.println(peer.getName() +" boot at time: "
                         +peer.getTime())
}
```

COSMOS

COntext entitieS coMpositiOn and Sharing

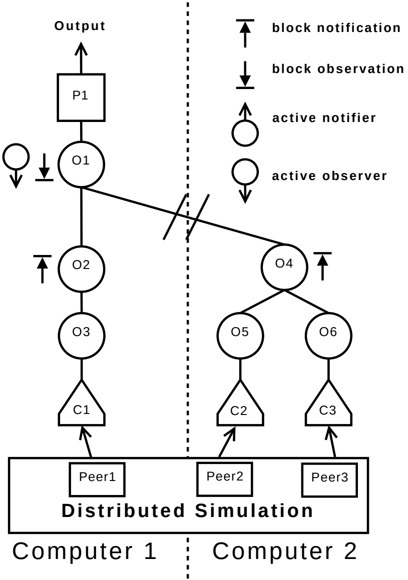
- Component-based framework for managing context data in ubiquitous applications
- Data processing built as a graph of processing nodes
- 3 COSMOS entities: collector, processor, policy
- Based on the Fractal component framework
  - Fractal ADL allows composition by inheritance and overloading
  - Fractal-BF turns components into services

COSMOS

COntext entitieS coMpositiOn and Sharing

- COSMOS context node
  - Hierarchical, with sharing
  - Parameterized
    - Passive or active
    - Observation or notification
    - Blocking or not
  - Input: message(s)
  - Output: compute new message
  - Message can contain sub-messages
  - Message chunks are typed
    - Ensure compatibility between context node

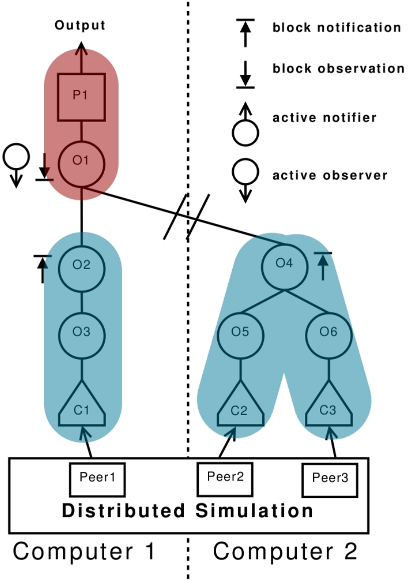
Live processing of data



Live processing of data

- COSMOS data processors and policies
  - Live analysis
    - Reduce bandwidth and memory overhead
  - Logging
    - Scave (OMNet++ post-processing tool)
  - Take into account the real topology of the simulation application and optimize the data flow

Composition



Composition

- Keep it simple
  - $\Longrightarrow$ Easier to manage and maintain
  - $\Longrightarrow$ More chance of reuse
- But build complex composition easily

Real experiments processing

- COSMOS is used for context observation in smart environments
  - We succesfully use COSMOS for instrumentation and data processing in simulation
- Apply the same data processing on real experiment and simulation
  - Validation of simulation results
  - Sharing data processing $\Longrightarrow$ more confidence in validation results

Conclusion on OSIF

- Separation of concerns
  - Favor reuse of models
  - Favor comparisons across simulators and platforms
- Live processing
  - Save disk space / bandwidth
- Composition
  - Build / manage / maintain simple instrumentation and data processing
  - Reuse data processing
  - Build complex data processing by composition of processors
- Apply data processing on real experiment
  - Reuse data processing
  - Validate simulation
  - Increase confidence

Questions ?

Thank you for your attention