Un mois en 10 minutes

William Waites
wwaites@tardis.ed.ac.uk

University of Edinburgh

Integrative Cell Models
Lorentz Centre, Leiden
January 29th, 2015

School of informatics

lfcs Laboratory for Foundations of Computer Science

MOdule Integration Simulator: Motivation

- Observations on Karr, Sanghvi, Macklin, *et al.* 2012
  - Modularisation good software engineering practice
  - Allocation strategy is problem-specific
- Modular models of complex systems aren't just for biology
  - Geophysics, finance & economics, population dynamics, epidemiology, μelectronics manufacturing
  - But interface standards (e.g. in climate) amongst modules amount to Fortran coding standards!
  - Software frameworks not directly useable for other problems
  - Processes often not directly useable in other models
- Allocation strategies[1] can be seen as integrators
  - Well studied in numerical analysis: general linear methods, splitting and composition methods, exponential and trigonometric integrators

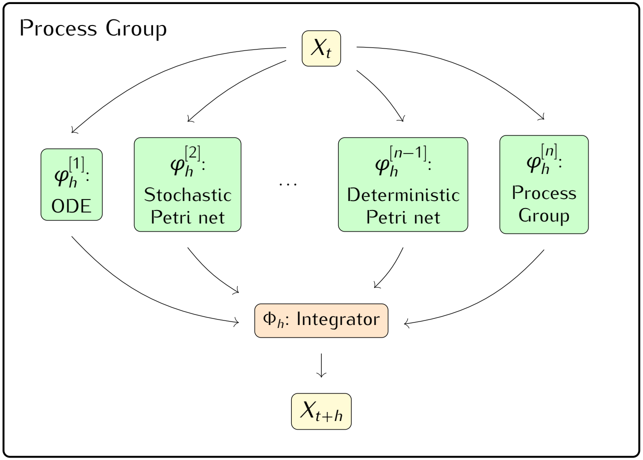
[1] in an ODE context at least

Can we make a better tool?

- Abstract out the *structure* of heterogeneous, modular systems
- Library of common classes of process
- The integrator[2] as a first-class citizen
- Convenient, natural syntax for describing problems
  - *Built into the programming environment for expressivity*
- Annotation for model re-use, composition and data provenance
- Portable, efficient i/o for multidimensional time-series

[2] a.k.a. resource allocator, a.k.a. scheduler

Processes and Process Groups


Process Group
X_t
$\varphi_h^{[1]}$: ODE
$\varphi_h^{[2]}$: Stochastic Petri net
…
$\varphi_h^{[n-1]}$: Deterministic Petri net
$\varphi_h^{[n]}$: Process Group
Φ_h: Integrator
X_{t+h}

Some Technical Details

- Strongly typed language – Scala

  ⇒ A decent type system has a steeper learning curve than e.g. Matlab or Python, but big payoff as compiler does a lot of the work for you

- NetCDF/HDF5 for I/O
  - Comes from earth/climate sciences for storing large multidimensional time-series
  - Very efficient and highly tuned
  - Everything can read it – Matlab, Octave, R, your favourite programming language
  - Can be self-documenting with embedded metadata
- RDF for machine-readable narrative description of the experiment and provenance of the output

  ⇒ and description of the interface for composition

Examples...

```
class GbKl extends StochasticReactionNetwork {
  annotate("skos:prefLabel", "Goldbeter-Koshland")
  val A = Species("gbkl:A")
  val B = Species("gbkl:B")
  val X = Species("gbkl:X")
  val Y = Species("gbkl:Y")

  reactions(
    A --> B catalysedBy X using MM(1, 1, 1),
    B --> A catalysedBy Y using MM(1, 1, 1)
  )
}
```

Examples...

```
class PredatorPrey(alpha...) extends ODE
    with Rosenbrock {
  val x = Double("pp:x")
  val y = Double("pp:y")
  d(x) := x * (alpha - beta * y)
  d(y) := -y * (gamma - delta * x)
}
```

```
class Pendulum extends Hamiltonian with Rosenbrock {
  var q = Double("pend:q")
      annotate("skos:prefLabel", "Generalised Position")
  var p = Double("pend:p")
      annotate("skos:prefLabel", "Generalised Momentum")
  H(q)(p) := pow(p,2)/2 + g*(1-cos(q))
}
```

Examples...

```
class MillarModel extends Model {
  // annotations, you get the idea
  val process = new ProcessGroup {
    scheduler = new CompositionScheduler(1.00)
  }
  process += new DaylightClock()
  process += new Photoperiodism()
  process += new Photothermal()
  process += new CarbonDynamic()
  process += new FunctionalStructural()

}
```

Process/Flow Types

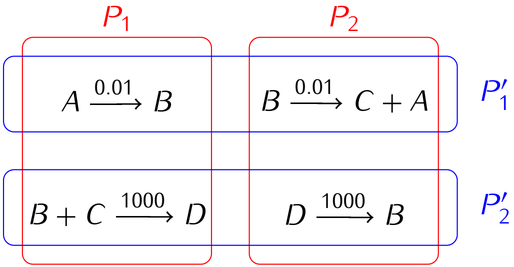
- Ordinary Differential Equations (choice of methods, RK4, Adams-Moulton, Rosenbrock, etc.)
- Deterministic Reaction Networks $\Rightarrow$ ODE
- Stochastic Reaction Networks $\Rightarrow$ Gillespie
- Classical Hamiltonian $\Rightarrow$ ODE
- Linear Optimisation (w/ non-linear objective)
- Kappa (ComingSoon!)
- SBML $\rightarrow$ Reaction Networks
- Recurrence Relations
- Process Group
- Ad-hoc Algorithms

Integrators or Schedulers

Composition	$X_{t+h} = \varphi_h^{[n]} \circ \varphi_h^{[n-1]} \circ \cdots \circ \varphi_h^{[2]} \circ \varphi_h^{[1]} \circ X_t$
Naive	Like **Composition** with random shuffling
Symmetric	$X_{t+h} = \varphi_{\frac{h}{2}}^{[3]} \circ \varphi_{\frac{h}{2}}^{[2]} \circ \varphi_h^{[1]} \circ \varphi_{\frac{h}{2}}^{[2]} \circ \varphi_{\frac{h}{2}}^{[3]} \circ X_t$
Weisse	Adaptive time-step method where step is adjusted according to relative change
Kick	Predictor-corrector method where each process runs for h and is corrected using $\partial\Phi_h/\partial X$
AdaptiveKick	Variant of **Kick** with time-step adjusted using the **Weisse** technique

Observation

The focus on modularity in cell models has been, similar to software engineering, for the benefit of the programmer. The benefits are obvious in terms of productivity and potential for re-use of modules. The resulting splitting or partitioning of the model, however, is unlikely to be optimal for computing the evolution of the system.



Work in Progress

- Transformation: automate decomposition and re-modularisation to suit available integration techniques
  - Strongly typed languages help with this *a lot*
  - Use this at run-time – “adaptive graining” and process migration
- Automate *reading* of metadata to check and adapt processes for composition

https://edinburgh-rbm.github.io/

Acknowledgements

N. Behr, D. Bucher, M. Cavaliere, V. Danos, I. Garnier, T. Heindel, **R. Honorato-Zimmer**, S. Jaramillo Riveri, G. Terradot, A. Weiße, **J. Wilson-Kanamori**

Fin

This project has received funding from the European Unions Seventh Framework Programme for research, technological development and demonstration under grant agreement no. #320823.

School of informatics

lfcs Laboratory for Foundations of Computer Science