Semi-formal models to support program development: autonomic management within component based parallel and distributed programming

Marco Danelutto

Dept. Computer Science - Univ. Pisa & CoreGRID Programming model Institute

FMCO 2008

Sophia Antipolis - October 2008

Contents

- Introduction
  - Functional vs. non-functional concerns
  - Autonomic management of non-functional concerns
- "Semi-formal" handling of non-functional concerns
  - ORC
- Use case
  - Performance tuning in stream parallel component compositions
- Conclusions

Contents

- Introduction
  - Functional vs. non-functional concerns
  - Autonomic management of non-functional concerns
- "Semi-formal" handling of non-functional concerns
  - ORC
- Use case
  - Performance tuning in stream parallel component compositions
- Conclusions

Functional & non functional concerns

- Functional
  - all aspects related to **what** is computed
- Non-functional
  - all aspects related to **how** the result is computed
- In parallel distributed programming
  - *functional*: the algorithm, the kind of parallel pattern used
  - *non-functional*: parallelism degree, load balance, fault tolerance, security, ...

Functional & non functional concerns

Application programmer concerns

- Functional
  - all aspects related to **what** is computed
- Non-functional
  - all aspects related to **how** the result is computed
- In parallel distributed programming
  - *functional*: the algorithm, the kind of parallel pattern used
  - *non-functional*: parallelism degree, load balance, fault tolerance, security, ...

Functional & non functional concerns

- Functional
  - all aspects related to **what** is computed
- Non-functional
  - all aspects related to **how** the result is computed
- In parallel distributed programming
  - *functional*: the algorithm, the kind of parallel pattern used
  - *non-functional*: parallelism degree, load balance, fault tolerance, security, ...

Application programmer concerns

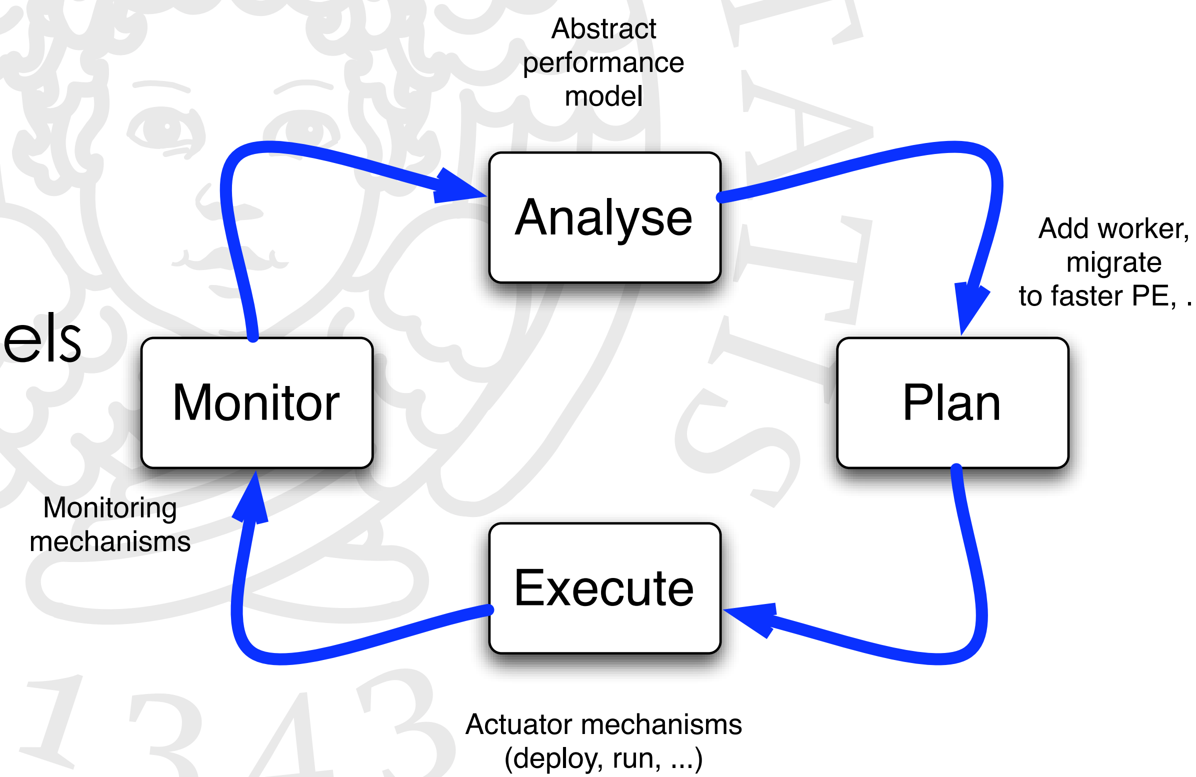
System programmer concerns

Autonomic management of non functional concerns

- Autonomic management
  - control loop: monitor → analyze → plan → execute
  - monitoring
    - mechanisms
  - analyzing
    - reference models
  - planning
    - strategies
  - executing
    - mechanisms

Autonomic management of non functional concerns

- Autonomic management
  - control loop: monitor → analyze → plan → execute
  - monitoring
    - mechanisms
  - analyzing
    - reference models
  - planning
    - strategies
  - executing
    - mechanisms



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Reactive autonomic management

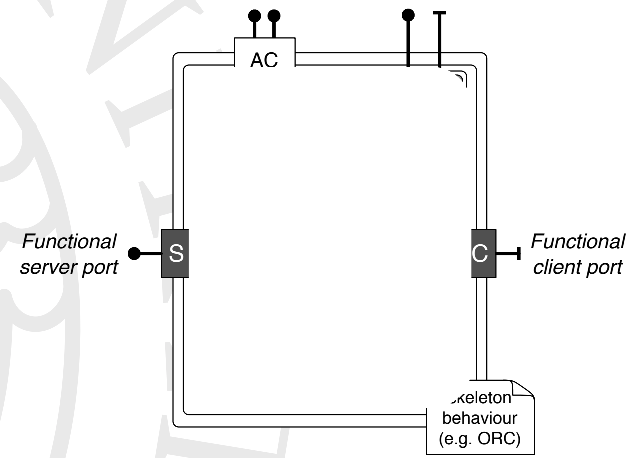
- Monitoring
  - non-invasive, immediate, effective
- Analysis
  - automatic, prioritized, extendible
- Planning
  - target architecture specific, optimized
- Execute
  - efficient, fast

Autonomic management in GCM

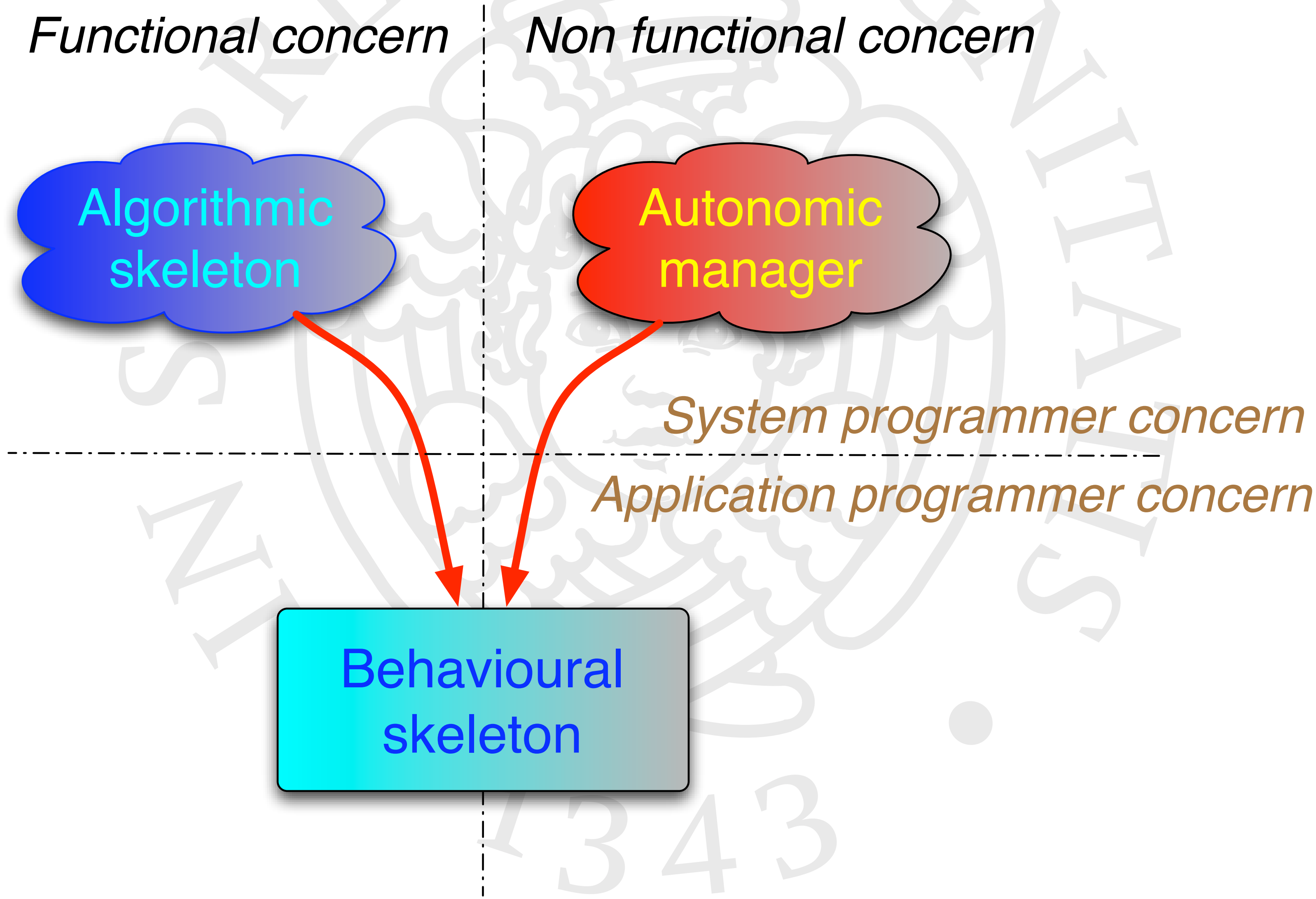
- Behavioural skeleton concept
  - co-design of
    - parallelism exploitation pattern
    - autonomic management
- Performance management
  - initial setup (parallelism degree)
  - optimization/tuning (load balancing, fault tolerance)
  - user driven (contract/SLA)

Autonomic management in GCM

- Behavioural skeleton concept
  - co-design of
    - parallelism exploitation pattern
    - autonomic management
- Performance management
  - initial setup (parallelism degree)
  - optimization/tuning (load balancing, fault tolerance)
  - user driven (contract/SLA)

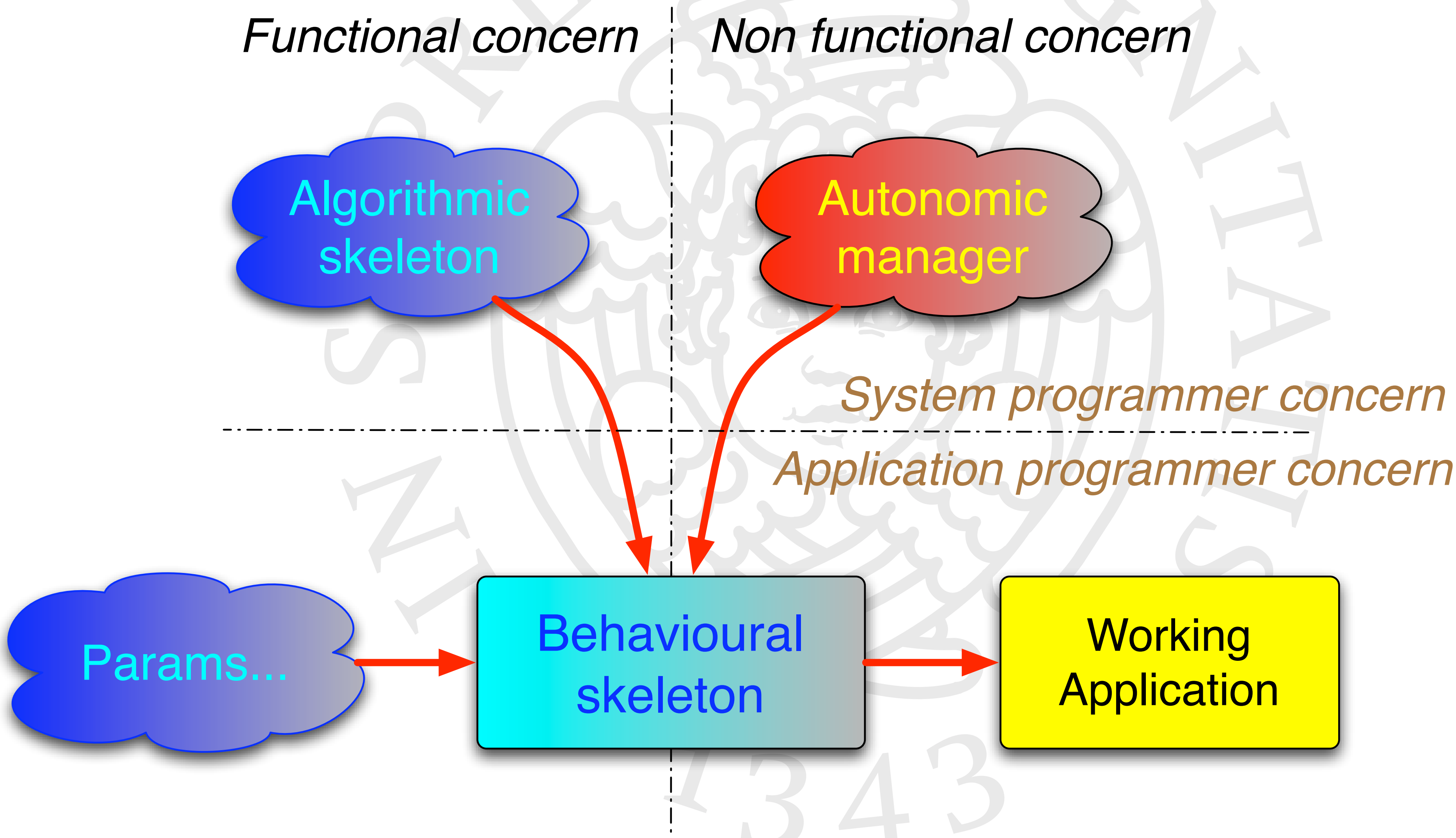


Behavioural skeleton



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

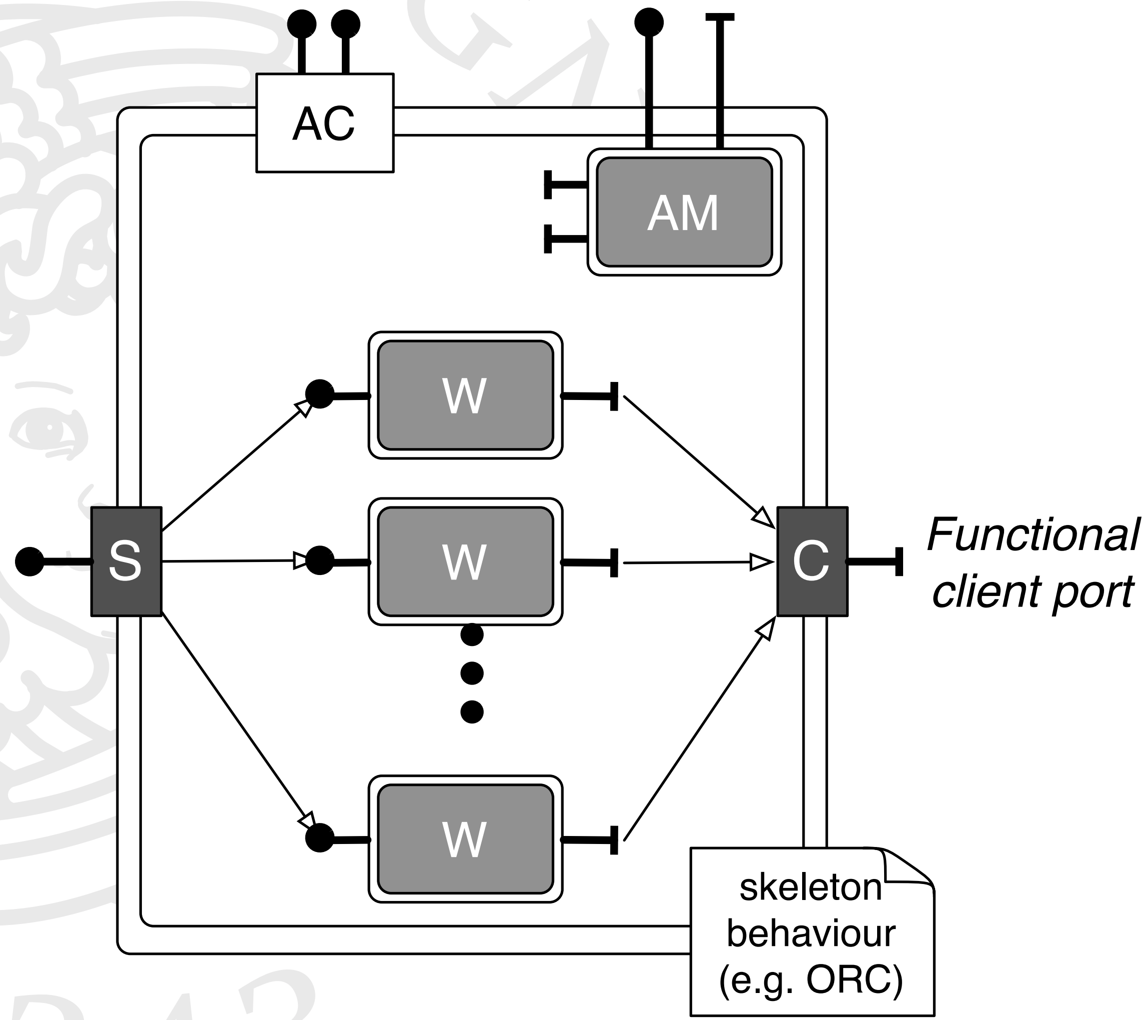
Behavioural skeleton



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Behavioural skeleton sample: functional replication

- Autonomic Controller
  implements passive actions
- Autonomic Manager
  manages performance tuning
- S port
  distributes input tasks to Ws
- C Port
  collects results from Ws
- Worker components
  compute results (functional)



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Rule based autonomic management

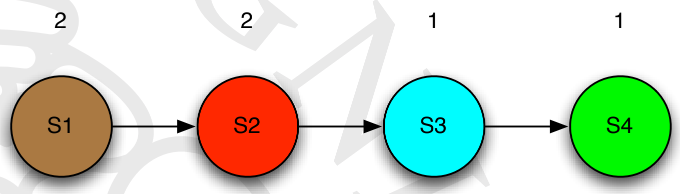
- triggering of actions
  - first order logic formulae over monitoring figures
- analysis
  - ordering / scheduling of the *fired* triggers
- planning/execute
  - sequence of mechanism invocation
- GCM uses JBoss rules (drools)
  - *rulename* salience *nn* when ... then ...

Already implemented (single pattern/ manager)

- parallelism degree adjustment
  - increase
    - suitable input pressure & unsatisfied contract
  - decrease
    - over satisfied contract
    - unsuitable input pressure
- fault tolerance
    - automatic recovery of faulty resources (muskel)

To be implemented (hierarchical pattern/ manager)

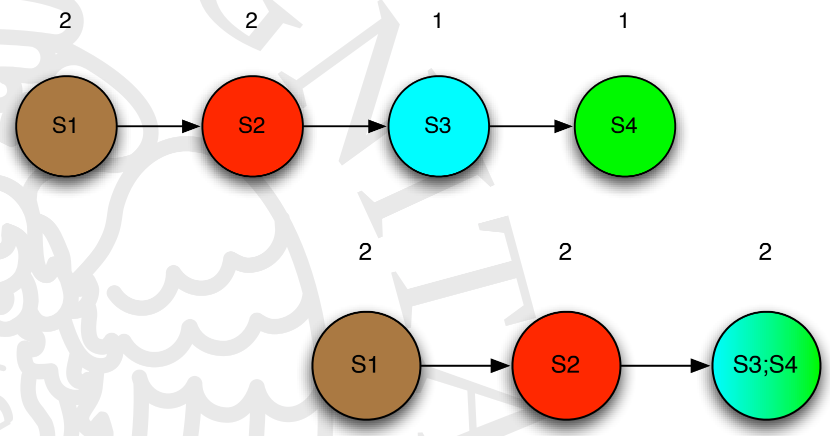
- change in (nesting of) parallelism exploitation patterns used
  - pipeline stage unbalance
    - stage collapsing
    - farm out stage
    - combination of collapsing and farming out



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

To be implemented (hierarchical pattern/ manager)

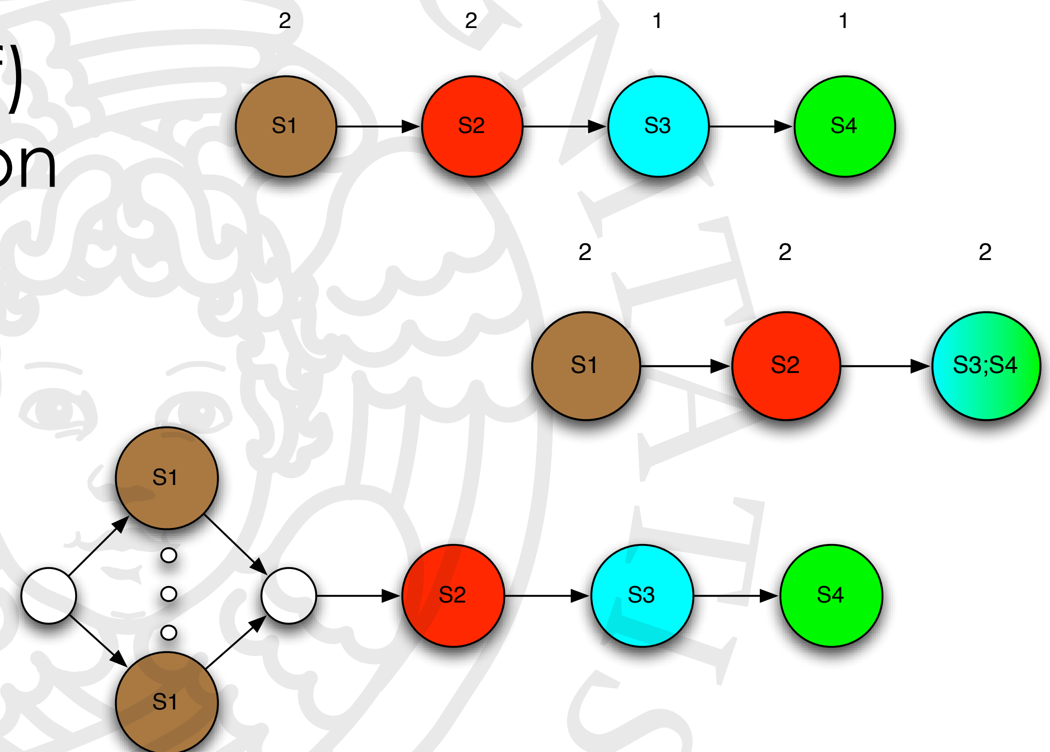
- change in (nesting of) parallelism exploitation patterns used
  - pipeline stage unbalance
    - stage collapsing
    - farm out stage
    - combination of collapsing and farming out



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

To be implemented (hierarchical pattern/ manager)

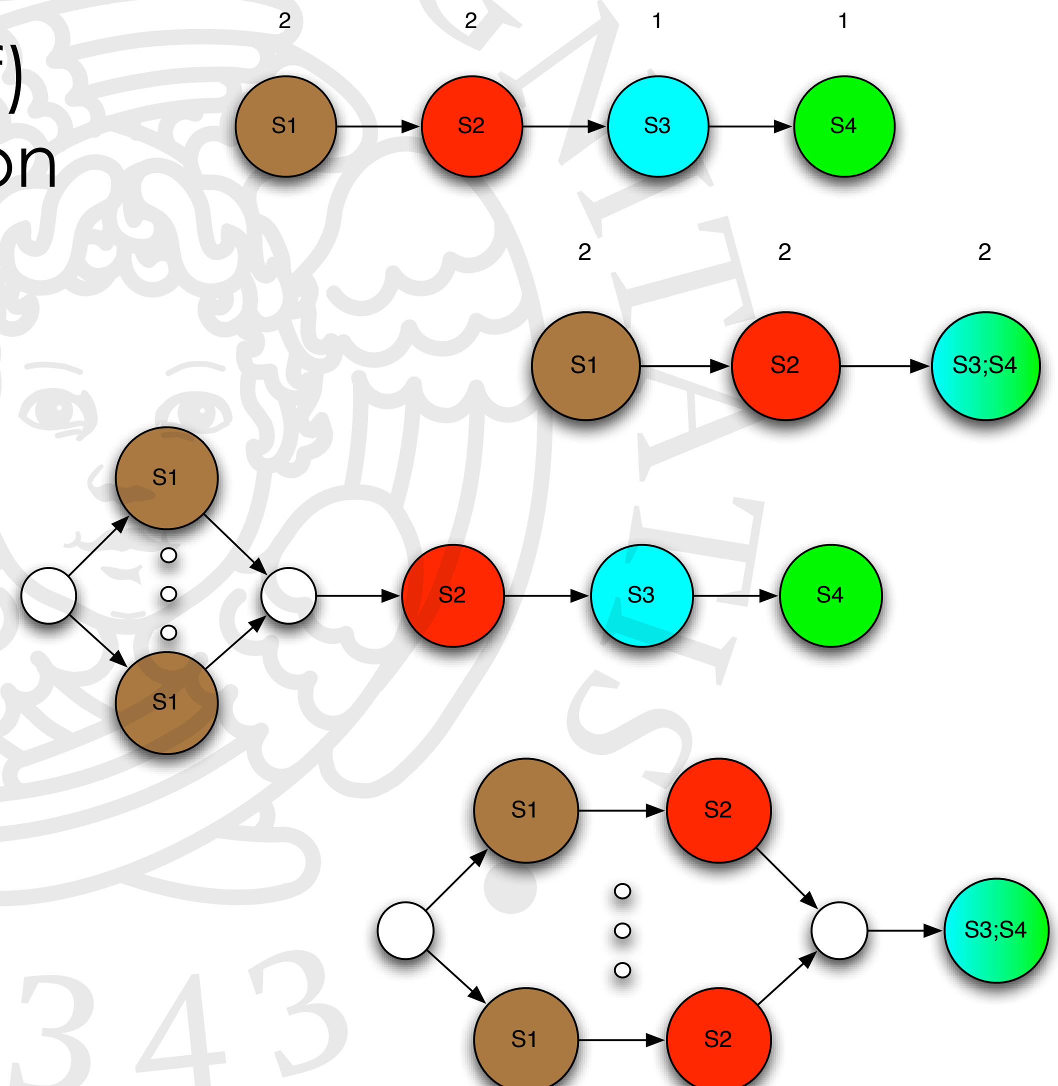
- change in (nesting of) parallelism exploitation patterns used
  - pipeline stage unbalance
    - stage collapsing
    - farm out stage
    - combination of collapsing and farming out



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

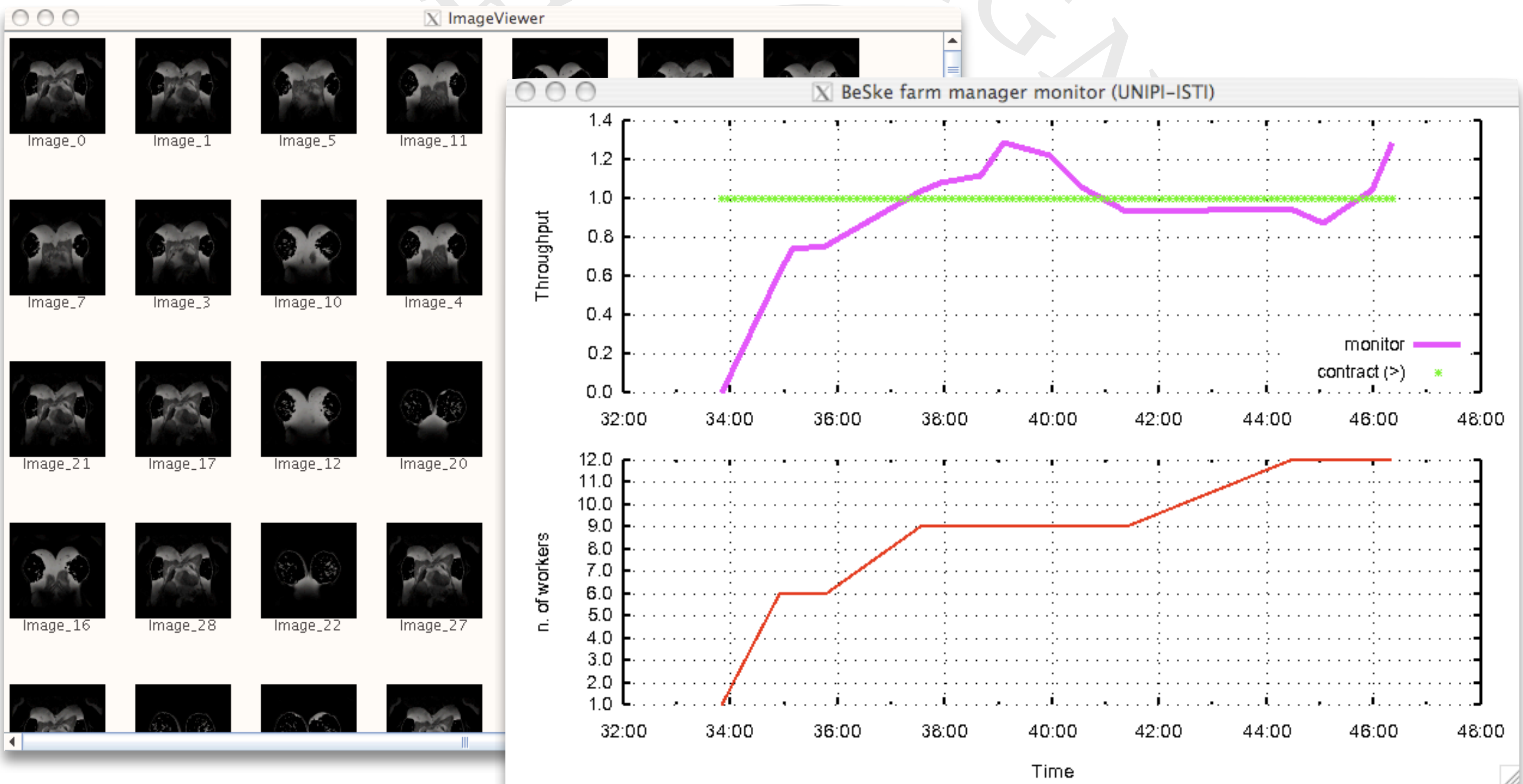
To be implemented (hierarchical pattern/ manager)

- change in (nesting of) parallelism exploitation patterns used
  - pipeline stage unbalance
    - stage collapsing
    - farm out stage
    - combination of collapsing and farming out



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Autonomic performance management @ work



M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Contents

- Introduction
  - Functional vs. non-functional concerns
  - Autonomic management of non-functional concerns
- "Semi-formal" handling of non-functional concerns
  - ORC
- Use case
  - Performance tuning in stream parallel component compositions
- Conclusions

Tools to support reasoning about autonomic management

- the two extremes
  - formal tools
    - consistent background needed
    - nice results demonstrated
      - possibly with limited scope
  - implementation
    - consistent background & ability needed
    - nice results “demonstrated”
      - possibly requiring a huge amount of time

Semi-formal tools

- preserve part of the knowledge typical of programmers
- preserve part of the techniques typical of formal tools
- e.g.
  - a framework
    - suitable to model interesting properties of a (distributed/parallel) program
    - synthetic
    - supporting program transformations

Orc Language Project

Home Download Documentation Research Community

Orc is ...

A novel language for distributed and concurrent programming. Orc provides uniform access to computational services, including distributed communication and data manipulation, through ***sites***. Using three simple concurrency primitives, the programmer ***orchestrates*** the invocation of sites to achieve a goal, while managing timeouts, priorities, and failures.

- Introduced by Misra and Cook in early '00
- provides primitive *combinators* for parallelism and non determinism (asymmetric parallelism)
- look&feel close to a programming language

What is it for?

- Concurrent and distributed programming
- Workflows (business process automation)
- Discrete event simulation
- Web service mashups

ORC in a slide

- site : local or remote (unreliable) unit of computation
- combinators
  - *a | b* site or expression a and b evaluated in parallel
  - a >> b (or a > x > b) a evaluated first, then b
  - *f(x) where x:∈ (a | b)* a,b and f started, as soon as either a or b produce a value, it is bound to x and (a | b) is terminated
    (new syntax *f(x)*<x<*(a | b)*)
- Functions
  - def f(param) = ...
- predefined sites + channels
  - if, Rtimer, +, -, ... , ch.get(), ch.put(x)

most/all the abstractions needed are there

Sample usage of ORC

- Reverse engineering (modelling) of muskel, a full Java/RMI skeleton library maintained at University of Pisa

$$
\begin{array}{l}
system(pgm, tasks, contract, G, t) \triangleq \\
\qquad taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t) \\
discovery(G, pgm, t) \triangleq (|_{g \in G}\ (\ \text{if } remw \neq \text{false} \gg rworkerpool.add(remw) \\
\qquad\qquad \text{where } remw :\in \\
\qquad\qquad\quad (\ \ g.can\_execute(pgm) \ \mid Rtimer(t) \gg let(false)\ ) \\
\qquad\qquad) \\
\qquad) \gg discovery(G, pgm, t) \\
manager(pgm, contract, t) \triangleq \\
\quad |i : 1 \leq i \leq contract : (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t)) \\
\quad \mid monitor \\
ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk > \\
\quad (\ \ \text{if } valid \ \gg\ resultpool.add(r) \ \gg\ ctrlthread_i(pgm, remw, t) \\
\quad \mid \text{if } \neg valid \ \gg\ (\ taskpool.add(tk) \\
\qquad\qquad \mid alarm.put(i) \ \gg\ c_i.get > w > ctrlthread_i(pgm, w, t) \\
\qquad\qquad) \\
\quad) \\
\qquad \text{where } (valid, r) :\in \\
\qquad\quad (\ \ remw(pgm, tk) > r > let(true, r) \ \mid\ Rtimer(t) \ \gg\ let(false, 0)\ \ ) \\
monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw) \\
\qquad \gg\ monitor
\end{array}
$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample usage of ORC

- Reverse engineering (modelling) of muskel, a full Java/RMI skeleton library maintained at University of Pisa

$$
\begin{aligned}
& system(pgm, tasks, contract, G, t) \triangleq \\
& \quad taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t) \\
& discovery(G, pgm, t) \triangleq (|_{g \in G}\ (\ \text{if } remw \neq \text{false} \gg rworkerpool.add(remw) \\
& \qquad\qquad \text{where } remw :\in \\
& \qquad\qquad\quad (\ \ g.can\_execute(pgm)\ \mid Rtimer(t) \gg let(false)\ ) \\
& \qquad\qquad) \\
& \qquad) \gg discovery(G, pgm, t) \\
& manager(pgm, contract, t) \triangleq \\
& \quad |i : 1 \leq i \leq contract : (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t)) \\
& \quad \mid monitor \\
& ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk > \\
& \quad (\ \ \text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t) \\
& \quad \mid \text{if } \neg valid \gg (\ taskpool.add(tk) \\
& \qquad\qquad \mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t) \\
& \qquad\qquad) \\
& \quad) \\
& \quad\quad \text{where } (valid, r) :\in \\
& \quad\quad\quad (\ \ remw(pgm, tk) > r > let(true, r)\ \mid\ Rtimer(t) \gg let(false, 0)\ \ ) \\
& monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw) \\
& \qquad\qquad \gg monitor
\end{aligned}
$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample usage of ORC

Autonomic management

- Reverse engineering (modelling) of muskel, a full Java/RMI skeleton library maintained at University of Pisa

$$
\begin{aligned}
& system(pgm, tasks, contract, G, t) \triangleq \\
& \qquad taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t) \\
& discovery(G, pgm, t) \triangleq (|_{g \in G} \; (\text{ if } remw \neq \text{false} \gg rworkerpool.add(remw) \\
& \qquad\qquad \text{where } remw :\in \\
& \qquad\qquad\qquad (\;\; g.can\_execute(pgm) \;\; \mid Rtimer(t) \gg let(false) \;) \\
& \qquad\qquad) \\
& \qquad) \gg discovery(G, pgm, t) \\
& manager(pgm, contract, t) \triangleq \\
& \quad |i : 1 \leq i \leq contract : (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t)) \\
& \quad \mid monitor \\
& ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk > \\
& \quad (\;\; \text{if } valid \;\gg\; resultpool.add(r) \;\gg\; ctrlthread_i(pgm, remw, t) \\
& \quad\; \mid \text{if } \neg valid \;\gg\; (\;\; taskpool.add(tk) \\
& \qquad\qquad\qquad \mid alarm.put(i) \;\gg\; c_i.get > w > ctrlthread_i(pgm, w, t) \\
& \qquad\qquad\quad) \\
& \quad) \\
& \qquad \text{where } (valid, r) :\in \\
& \qquad\quad (\;\; remw(pgm, tk) > r > let(true, r) \;\mid\; Rtimer(t) \;\gg\; let(false, 0) \;\;) \\
& monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw) \\
& \qquad\quad \gg \; monitor
\end{aligned}
$$

monitoring

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming*, FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample usage of ORC

- Reverse engineering (modelling) of muskel, a full Java/RMI skeleton library maintained at University of Pisa

$$
\begin{array}{l}
system(pgm, tasks, contract, G, t) \triangleq \\
\quad taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t) \\[4pt]
discovery(G, pgm, t) \triangleq (|_{g \in G}\ (\ \text{if } remw \neq \text{false} \gg rworkerpool.add(remw) \\
\qquad\qquad \text{where } remw :\in \\
\qquad\qquad\quad (\ \ g.can\_execute(pgm)\ \mid Rtimer(t) \gg let(false)\ ) \\
\qquad\qquad) \\
\qquad)\gg discovery(G, pgm, t) \\[4pt]
manager(pgm, contract, t) \triangleq \\
\quad |i : 1 \leq i \leq contract : (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t)) \\
\quad \mid monitor \\[4pt]
ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk > \\
\quad (\ \ \text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t) \\
\quad \mid \text{if } \neg valid \gg (\ taskpool.add(tk) \\
\qquad\qquad \mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t) \\
\qquad\qquad) \\
\quad) \\
\qquad \text{where } (valid, r) :\in \\
\qquad\quad (\ \ remw(pgm, tk) > r > let(true, r)\ \mid\ Rtimer(t) \gg let(false, 0)\ ) \\[4pt]
monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw) \\
\qquad \gg monitor
\end{array}
$$

triggering

monitoring

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Core GRID GridCOMP Effective Components for the Grids

Autonomic management

Sample usage of ORC

- Reverse engineering (modelling) of muskel, a full Java/RMI skeleton library maintained at University of Pisa

$$
\begin{aligned}
& system(pgm, tasks, contract, G, t) \triangleq \\
& \quad taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t) \\
& discovery(G, pgm, t) \triangleq (|_{g \in G} \; (\text{ if } remw \neq \text{false} \gg rworkerpool.add(remw) \\
& \qquad\qquad \text{where } remw :\in \\
& \qquad\qquad\quad (\;\; g.can\_execute(pgm) \;\mid Rtimer(t) \gg let(false) \,) \\
& \qquad\qquad) \\
& \qquad) \gg discovery(G, pgm, t) \\
& manager(pgm, contract, t) \triangleq \\
& \quad |i : 1 \leq i \leq contract : (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t)) \\
& \quad \mid monitor \\
& ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk > \\
& \quad (\;\text{ if } valid \gg \; resultpool.add(r) \;\gg\; ctrlthread_i(pgm, remw, t) \\
& \quad \mid \text{if } \neg valid \;\gg\; (\; taskpool.add(tk) \\
& \qquad\qquad \mid alarm.put(i) \;\gg\; c_i.get > w > ctrlthread_i(pgm, w, t) \\
& \qquad\qquad) \\
& \quad) \\
& \qquad \text{where } (valid, r) :\in \\
& \qquad\quad (\;\; remw(pgm, tk) > r > let(true, r) \;\mid\; Rtimer(t) \;\gg\; let(false, 0) \;\;) \\
& monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw) \\
& \qquad \gg \; monitor
\end{aligned}
$$

triggering

plan
execute

monitoring

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Autonomic management

Sample usage of ORC

- Reverse engin… odelling) of muskel, a full Java/RMI … rary maintained at University of Pisa …

M. Aldinucci, M. Danelutto, and P. Kilpatrick. Management in distributed systems: a semi-formal approach. In A. M. Kermarrec, T. Priol, and L. Bougé, editors, Proc. of 13th Intl. Euro-Par 2007 Parallel Processing, volume 4641 of LNCS, pages 651—661, Rennes, France, August 2007. Springer

$$\ldots contract, G, t) \triangleq$$
$$\ldots .add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t)$$

$$\ldots ery(G, pgm, t) \triangleq (|_{g \in G} \; (\text{ if } remw \neq \text{false} \gg rworkerpool.add(remw)$$
$$\text{where } remw :\in$$
$$(\; g.can\_execute(pgm) \mid Rtimer(t) \gg let(false)\;)$$
$$)$$
$$) \gg discovery(G, pgm, t)$$

$$manager(pgm, contract, t) \triangleq$$
$$|i : 1 \leq i \leq contract : (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t))$$
$$\mid monitor$$

$$ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk >$$
$$(\;\text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t)$$
$$\mid \text{if } \neg valid \gg (\; taskpool.add(tk)$$
$$\mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t)$$
$$)$$
$$)$$
$$\text{where } (valid, r) :\in$$
$$(\; remw(pgm, tk) > r > let(true, r) \mid Rtimer(t) \gg let(false, 0)\;)$$

$$monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw)$$
$$\gg monitor$$

triggering

plan

execute

monitoring

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Contents

- Introduction
  - Functional vs. non-functional concerns
  - Autonomic management of non-functional concerns
- "Semi-formal" handling of non-functional concerns
  - ORC
- Use case
  - Performance tuning in stream parallel component compositions
- Conclusions

ORC for autonomic management

- semi formal derivation / proof of rewriting rules
- e.g. pipe(f,g) = pipe(farm(f),g)
  - simple example, technique to be used for more complex derivations
  - first step: formalization of skeletons
  - second step: semi formal processing of ORC expressions

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Formalization of skeletons

$$\begin{aligned}
\text{pipe}(A, B, ch_{in}, ch_{out}) &= \text{stage}(A, ch_{in}, ch_{new}) \\
&\quad \mid \text{stage}(B, ch_{new}, ch_{out}) \\
\text{stage}(A, ch_{in}, ch_{out}) &= ch_{in}.get() > task > A(task) > \\
&\quad result > ch_{out}.put(result) >> \\
&\quad \text{stage}(A, ch_{in}, ch_{out}) \\
\text{farm}(W, nw, c_{in}, ch_{out}) &= \mid \; i = 1, nw \; : \; \text{Worker}_i(W, c_{in}, ch_{out}) \\
\text{Worker}(W, c_{in}, ch_{out}) &= ch_{in}.get() > task > W(task) > \\
&\quad result > ch_{out}.put(result) >> \\
&\quad Worker(W, c_{in}, ch_{out})
\end{aligned}$$

Semi-formal processing of ORC expressions

- e.g. exploit semantics for channels
  - matching get/put pair collapsing
  - in actual traces
  - (R free for x !)

$$\frac{(a > x > ch.put(x) > R) \mid (\ldots >> ch.get() > y > S)}{R \mid \ldots >> a > y > S}$$

Sample semi-formal reasoning

$$\text{pipe}(A, B, c_1, c_3) = \text{stage}(A, c_1, c_2) \mid \text{stage}(B, c_2, c_3)$$

$$\text{seq}(A, B, ch_{in}, ch_{out}) = c_{in}.get() > x > A(x) > y > B(y) > z > ch_{out}.put(z)$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming*, FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample semi-formal reasoning

$$\text{pipe}(A, B, c_1, c_3) = \text{stage}(A, c_1, c_2) \mid \text{stage}(B, c_2, c_3)$$

$$\text{seq}(A, B, ch_{in}, ch_{out}) = c_{in}.get() > x > A(x) > y > B(y) > z > ch_{out}.put(z)$$

$$c_1.get() > t > A(t) > y > c_2.put(y) >> \text{stage}(A, c_1, c_2)$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample semi-formal reasoning

$$\text{pipe}(A, B, c_1, c_3) = \text{stage}(A, c_1, c_2) \mid \text{stage}(B, c_2, c_3)$$

$$\text{seq}(A, B, ch_{in}, ch_{out}) = c_{in}.get() > x > A(x) > y > B(y) > z > ch_{out}.put(z)$$

$$c_1.get() > t > A(t) > y > c_2.put(y) >> \text{stage}(A, c_1, c_2)$$

$$c_2.get() > t > B(t) > y > c_3.put(y) >> \text{stage}(B, c_2, c_3)$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample semi-formal reasoning

$$\mathsf{pipe}(A, B, c_1, c_3) = \mathsf{stage}(A, c_1, c_2) \mid \mathsf{stage}(B, c_2, c_3)$$

$$\mathsf{seq}(A, B, ch_{in}, ch_{out}) = c_{in}.get() > x > A(x) > y > B(y) > z > ch_{out}.put(z)$$

$$c_1.get() > t > A(t) > y > c_2.put(y) >> \mathsf{stage}(A, c_1, c_2)$$

$$c_2.get() > t > B(t) > y > c_3.put(y) >> \mathsf{stage}(B, c_2, c_3)$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming*, FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample semi-formal reasoning

$$\mathsf{pipe}(A, B, c_1, c_3) = \mathsf{stage}(A, c_1, c_2) \mid \mathsf{stage}(B, c_2, c_3)$$

$$\mathsf{seq}(A, B, ch_{in}, ch_{out}) = c_{in}.get() > x > A(x) > y > B(y) > z > ch_{out}.put(z)$$

$$c_1.get() > t > A(t) > y > c_2.put(y) >> \mathsf{stage}(A, c_1, c_2)$$

$$c_2.get() > t > B(t) > y > c_3.put(y) >> \mathsf{stage}(B, c_2, c_3)$$

$$c_1.get() > x > \mathsf{stage}(seq(A, B))(x) > y > c_3.put(y)$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Sample semi-formal reasoning

$$\text{pipe}(A, B, c_1, c_3) = \text{stage}(A, c_1, c_2) \mid \text{stage}(B, c_2, c_3)$$

$$\text{seq}(A, B, ch_{in}, ch_{out}) = c_{in}.get() > x > A(x) > y > B(y) > z > ch_{out}.put(z)$$

$$c_1.get() > t > A(t) > y > c_2.put(y) >> \text{stage}(A, c_1, c_2)$$

$$c_2.get() > t > B(t) > y > c_3.put(y) >> \text{stage}(B, c_2, c_3)$$

$$c_1.get() > x > \text{stage}(seq(A, B))(x) > y > c_3.put(y)$$

$$\text{pipe}(A, B, ch_{in}, ch_{out}) \leftrightarrow \text{seq}(A, B, ch_{in}, ch_{out})$$

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Design of autonomic managers

$$\text{BSekl}(Sk, Mgr, SLA) = Sk \mid Mgr(Sk, SLA)$$

$$\begin{aligned} Mgr(Sk, SLA) = \quad & distribute(Sk, SLA) > s > \\ & monitor(s) > m > \\ & analyse(s, m) > (b, p, v) > \\ & ((if(b) >> adapt(s, p) > s1 > \\ & \quad Mgr(s1, SLA)) \\ & \mid (if(\sim b) >> raise(v) > \\ & \quad Mgr(s, passiveMode(SLA))) \end{aligned}$$

Design of autonomic managers

$$\mathsf{BSekl}(Sk, Mgr, SLA) = Sk \mid Mgr(Sk, SLA)$$

$$\begin{aligned} Mgr(Sk, SLA) = \quad & distribute(Sk, SLA) > s > \\ & monitor(s) > m > \\ & analyse(s, m) > (b, p, v) > \\ & ((if(b) >> adapt(s, p) > s1 > \\ & \quad Mgr(s1, SLA)) \\ & \mid (if(\sim b) >> raise(v) > \\ & \quad Mgr(s, passiveMode(SLA))) \end{aligned}$$

Marco Aldinucci, Marco Danelutto, Peter Kilpatrick, Towards hierarchical management of autonomic components: a case study, Euromicro PDP 2009, IEEE Press

M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Adaptation in BS

$$
\begin{aligned}
&adapt(\mathsf{pipe}(A, pipe(B,C)), plan) = \\
&\quad (if(plan = collapseFirst) >> \mathsf{pipe}(\mathsf{seq}(A,B),C)) \\
&\quad |\ (if(plan == collapseLast) >> \mathsf{pipe}(A, \mathsf{seq}(B,C))) \\
&\quad |\ (if(plan == farmoutFirst) >> \mathsf{pipe}(\mathsf{farm}(A), \mathsf{pipe}(B,C))))
\end{aligned}
$$

- modelling of management *before* actual implementation

Adding metadata

- annotations on Orc code
  - modelling several non functional concerns
  - e.g. security, communication costs
- formal process deriving the aggregated metadata from primitive/elementary/ground one (synthesis) or primitive metadata from aggregated (analysis)

Sample metadata: process placement

- Analysis
  - placement annotations
  - policy managing nested skeleton annotations

$$\frac{placement(\mathsf{pipe}(A,B), loc(X)) \ \wedge \ distribPolicy(keep)}{placement(A, loc(X)) \ \wedge \ placement(B, loc(X))}$$

$$\frac{placement(\mathsf{pipe}(A,B), loc(X)) \ \wedge \ distribPolicy(distrib)}{placement(A, loc(fresh())) \ \wedge \ placement(B, loc(fresh()))}$$

Sample metadata usage

- skeleton program + placement metadata (includes support: channels, manager process(es), ...)
- communication cost

$$\frac{placement(ch.get(), loc(X) \wedge placement(ch, loc(Y))}{nonLocalCost(ch.get())}$$

$$\frac{placement(ch.get(), loc(X) \wedge placement(ch, loc(Y)}{nonLocalCost(ch.get())}$$

- automatic derivation of communication cost in typical traces of execution

Sample metadata usage

- skeleton program + placement metadata (… support: channels, manager process(…
- communication cost

$$\frac{placement(ch.g \ldots \; \ldots ement(ch, loc(Y))}{\ldots ost(ch.get())}$$

$$\frac{\ldots get(), loc(X) \wedge placement(ch, loc(Y)}{nonLocalCost(ch.get())}$$

Marco Aldinucci, Marco Danelutto and Peter Kilpatrick, A framework for prototyping and reasoning about grid systems, in: Parallel Computing: Architectures, Algorithms and Applications (Proc. of {PARCO 2007}, Jülich, Germany), John von Neumann Institute for Computing, 2007

- automatic derivation of communication cost in typical traces of execution

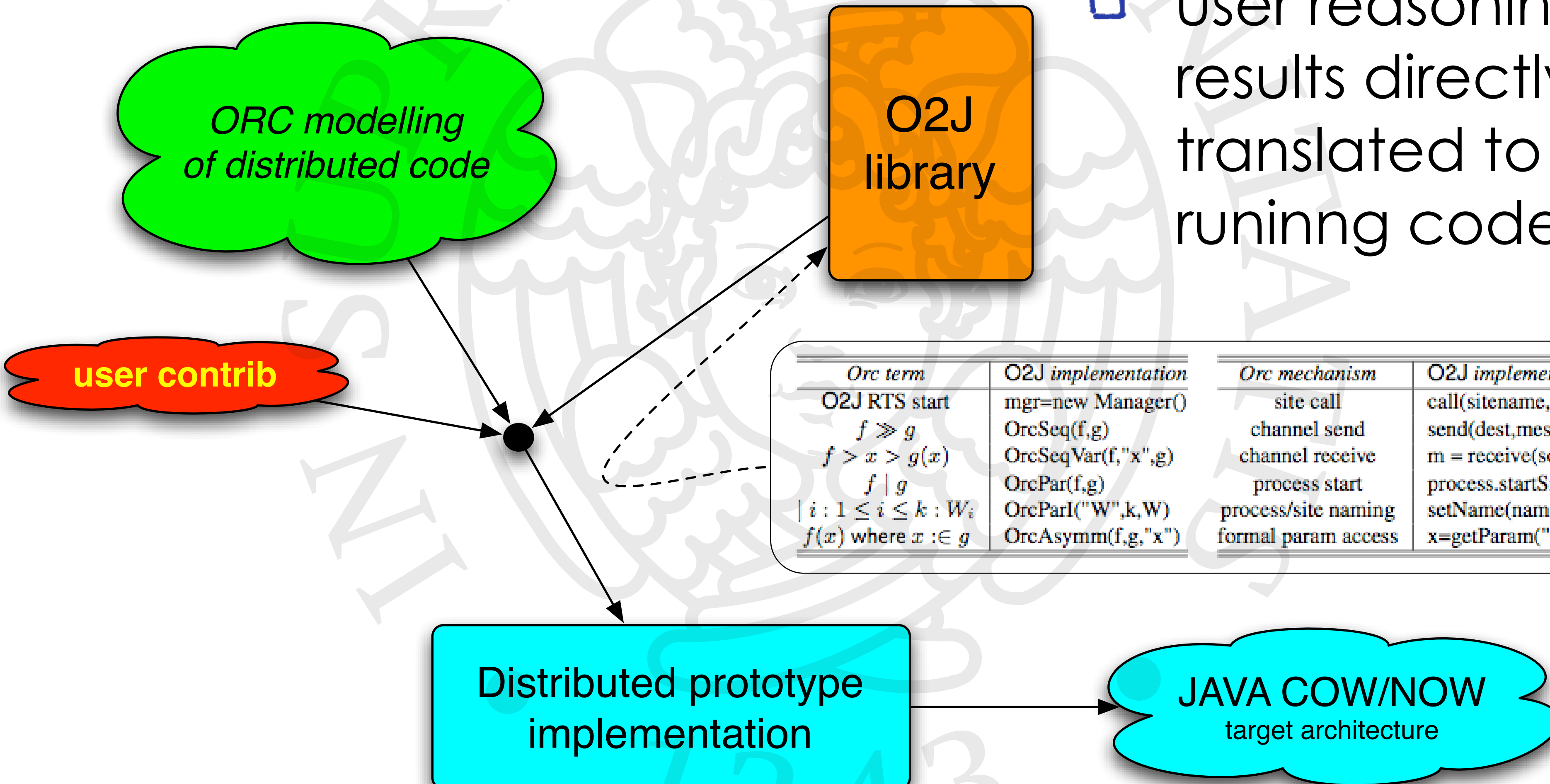
Sample metadata: security

- Synthesis

$$\frac{placement(A, loc(X)) \wedge insecure(X)}{insecure(\textsf{pipe}(A, \_)) \;\; insecure(\textsf{pipe}(\_, A))}$$

  - marking of root depending on the marks at leaves
- Usage:
  - node marking => securing code and data only when needed

An Orc based development framework



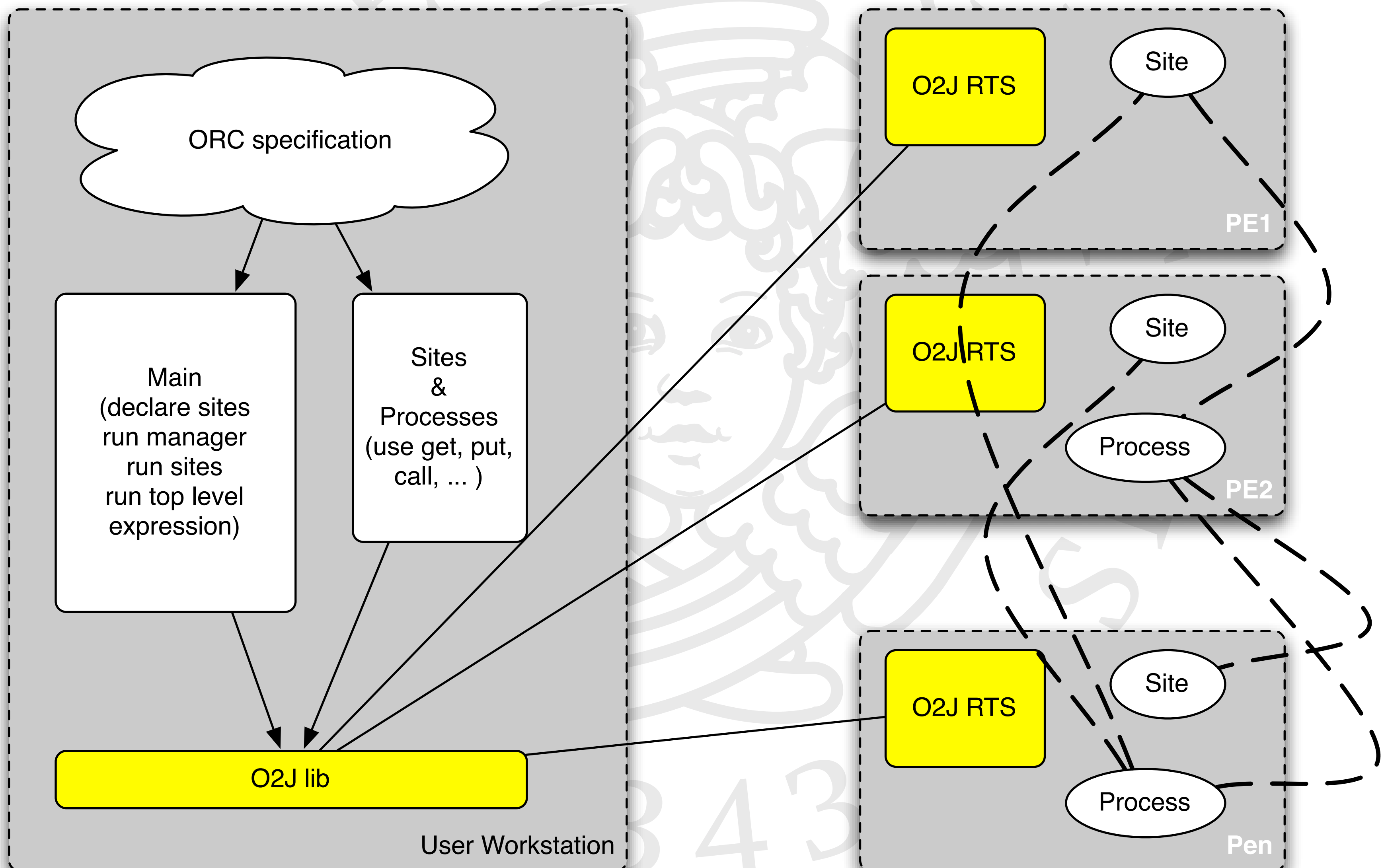
Orc term	O2J implementation
O2J RTS start	mgr=new Manager()
$f \gg g$	OrcSeq(f,g)
$f > x > g(x)$	OrcSeqVar(f,"x",g)
$f \mid g$	OrcPar(f,g)
$\mid i : 1 \leq i \leq k : W_i$	OrcParI("W",k,W)
$f(x)$ where $x :\in g$	OrcAsymm(f,g,"x")

Orc mechanism	O2J implementation
site call	call(sitename,message)
channel send	send(dest,message)
channel receive	m = receive(source)
process start	process.startSite()
process/site naming	setName(name)
formal param access	x=getParam("x")

- user reasoning results directly translated to runinng code

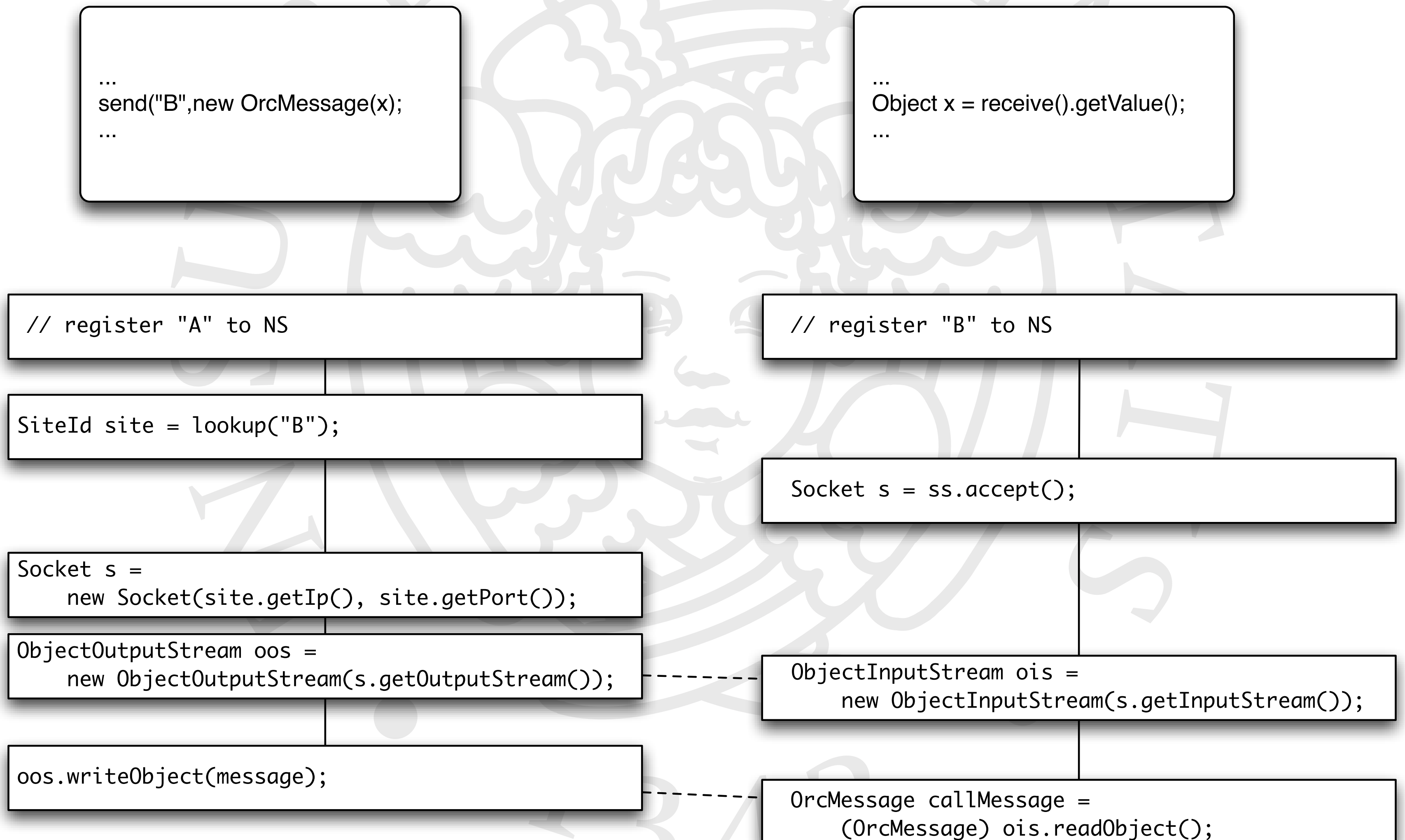
M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

O2J internals (abstract view)



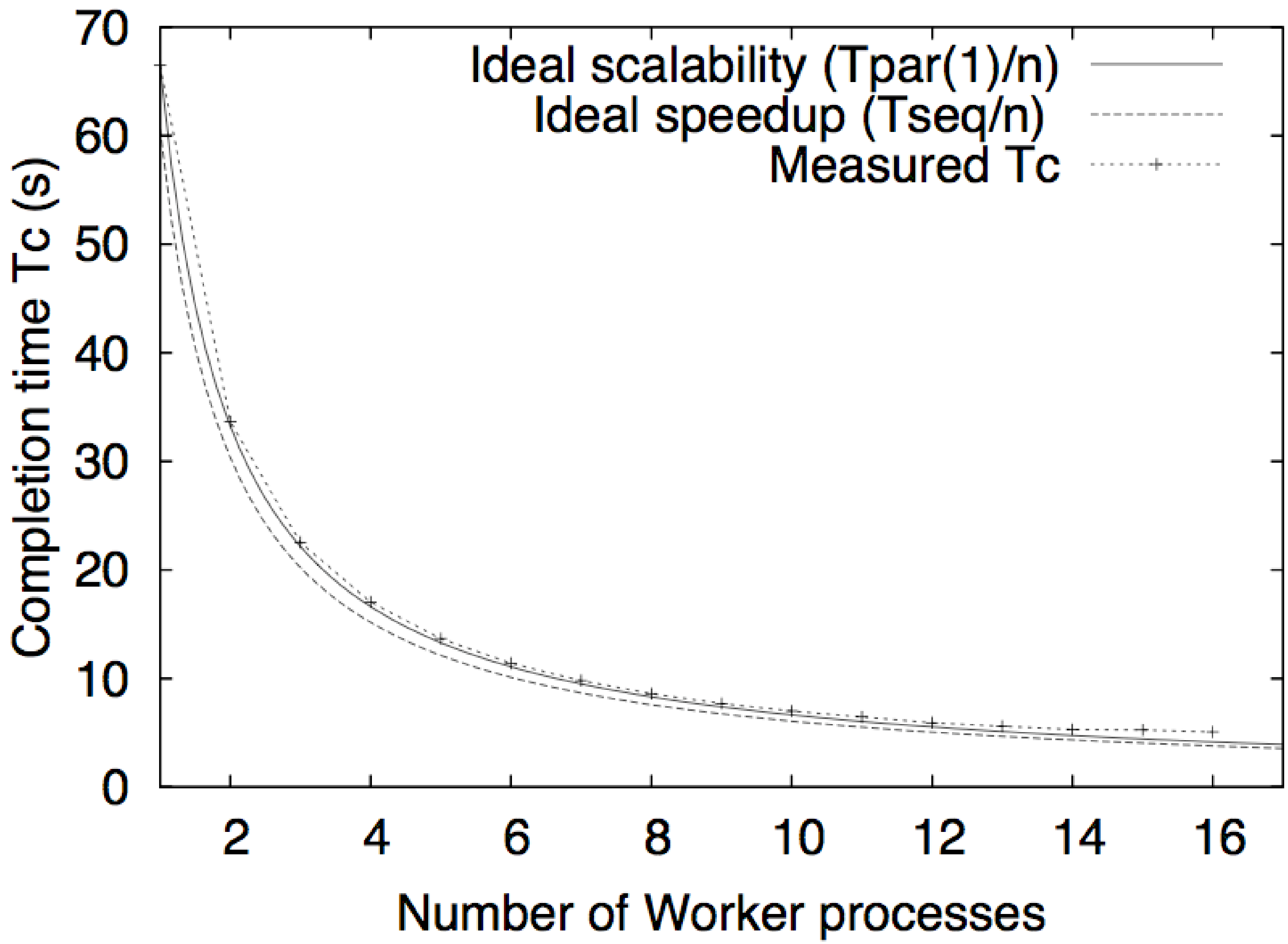
M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming, *FMCO 2008, Sophia Antipolis, October 22nd, 2008*

O2J internals (2)

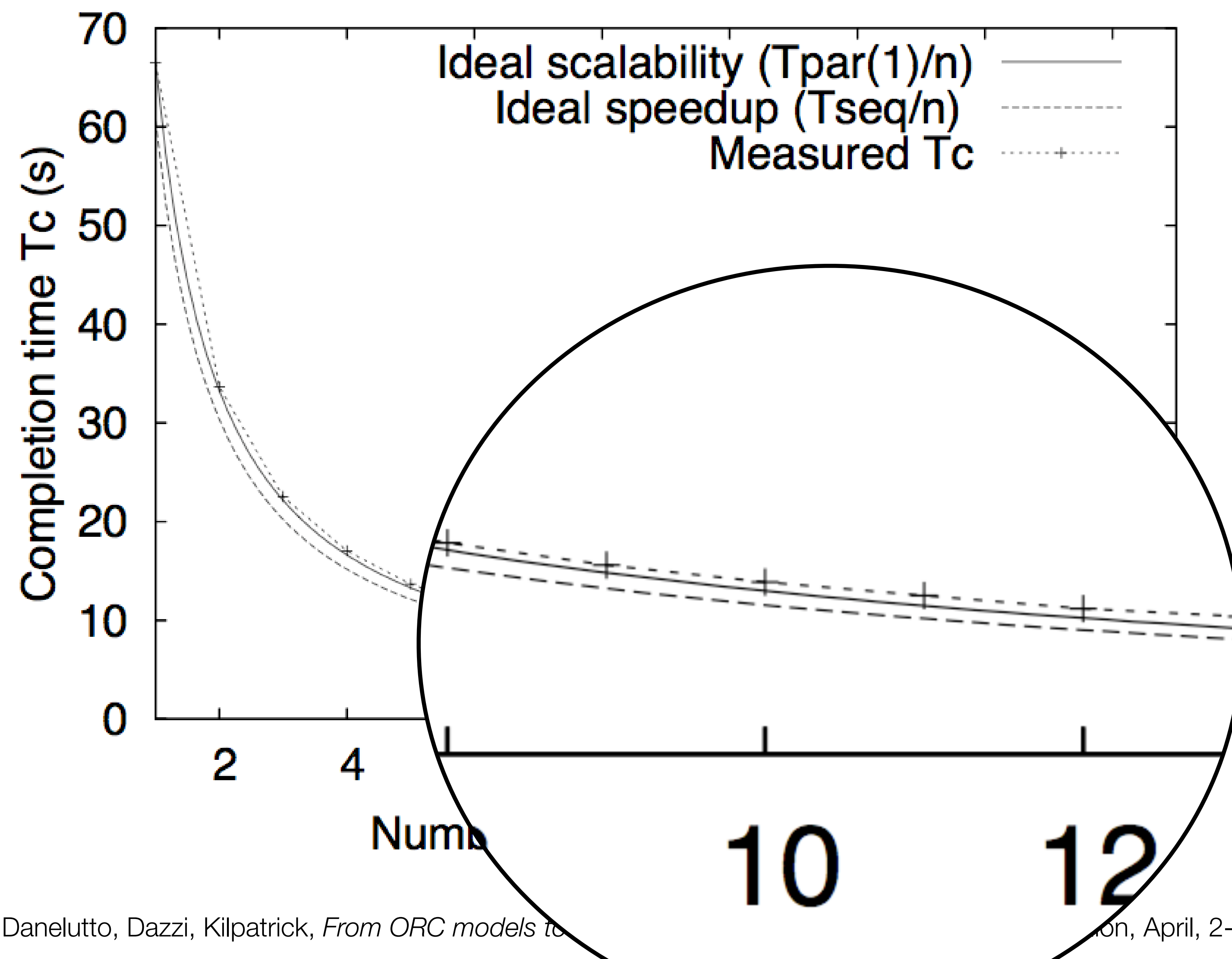


M. Danelutto, Semi-formal models to support program development: autonomic management within component based parallel and distributed programming*, FMCO 2008, Sophia Antipolis, October 22nd, 2008*

Experimental results



Experimental results



Conclusions

- Autonomic management of non functional features
  - a must
  - a complex task
- Semi formal modelling
  - provides insights and design hints
  - can be used to support reasoning
    - event stronger with proper metadata
- Experience in GCM / CoreGRID / GridCOMP

Thank you for your attention

Any questions ?