Components for grids
(an evolutionary process)

Marco Danelutto
Dept. of Computer Science Univ. of Pisa
&
Programming model Institute - CoreGRID
Outline

- Introduction
- EchoGrid Roadmap
- Key aspects in (grid) sw development
- Autonomic aspects in GCM
- Solving the Service / Component dicotomia
- Perspectives
Introduction

- For a given problem or applications, specification of the orchestration should be done using high-level abstractions having the following properties: intuitive so that non-expert programmers can use them, generic to handle a large spectrum of applications and parallelism should be implicit and fully hidden to the programmers. Moreover, to cope with the large scale and unreliable dimension of the grid, programming languages, providing these high-level abstractions, have to be associated with a distributed execution model to avoid any bottleneck.

Roadmap, page 6
Introduction

If we could efficiently handle the predicted complexity derived from the large-scale interactions between future Grid applications, we could conclude to the following visionary ideas that may be proved useful as a guide in the technological advance of this area:

1. Produce component model to support *autonomic software development* so as to improve adaptation and reusability at a high level.
2. Develop a unified component model, that will fit the needs of future industrial use cases as well as those of today’s Grid applications.
3. Clearer *separation of concerns regarding the functional and non-functional requirements* to ease the maintenance and the flexibility.
4. Standardize software certification models (such as the testing model for Grid software) at the component model level in order to improve software quality assurance.
5. The construction of reliable and scalable software systems require better behavioural guarantees. Elements towards this goal include: *more expressive specification formalisms*, *development environment providing “correct by construction” code*, *dynamic adaptation techniques*, etc.
6. Ensure QoS at the component level which will allow component applications to provide support SLA requirements.

Roadmap, pp 26
Key aspects in sw development

- Incremental design
  - separate test & debugging (functional & non functional)
  - “composability” (syntactic & semantic)
  - fundamental in the past (Unix!)
- Fast prototyping
  - ability to move from concept to code (tools, tools, tools)
- Interoperability
  - syntactic (easy), semantic (!?!?!)

Marco Danelutto Components for Grids: an evolutionary process EchoGrid meeting Athens, June 2008
Key aspects in grid sw development

- Resource management
  - initial mapping/scheduling,
  - adaptivity (fault tolerance)
- Fault tolerance
  - checkpointing, ...
- Security
  - active, passive
Component models
Component models

- Component = unit of deployment
  - functionality encapsulated much better than objects
- Programs = component assemblies
  - usually (hopefully): tested, third party components
- Composite components = components
  - more and more abstraction levels supported
Components & Grids
Components & Grids

- Deployment, life cycle, etc.
  - managed by the framework/tools
- Non functional aspects in component (assembly) execution
  - not in charge of the application programmer!
- Interoperability
  - guaranteed with major grid frameworks (WS)
Components & Grids

- Deployment, life cycle, etc.
  - managed by the framework/tools
- Non functional aspects in component (assembly) execution
  - not in charge of the application programmer!
- Interoperability
  - guaranteed with major grid frameworks (WS)
Autonomic domains
Autonomic domains

- performance tuning
  - resource management, parallelism degree management, subcomputation rearranging, ...
- fault tolerance
  - checkpoint and restart on failure symptoms, management of distributed checkpoints, ...
- security
  - security channel proxying, alternative communication media handling
GCM

- Hierarchical component composition
- Collective + data/stream ports
- Autonomic management of notable composite components
- XML based ADL
- Reference implementation in GridCOMP
- Fractal based, in ProActive
- Hierarchical component composition
- Collective + data/stream ports
- Autonomic management of notable composite components
- XML based ADL
- Reference implementation in GridCOMP
  - Fractal based, in ProActive
GCM

autonomic managers

- Combining skeleton technology with autonomic management of performance
- Behavioural skeletons (BS)
- In GridCOMP/CoreGRID:
  - functional replication BS (task farm, data parallel)
  - rule based autonomic managers
GCM

autonomic managers

- Combining skeleton technology with autonomic management of performance
- Behavioural skeletons (BS)
- In GridCOMP/CoreGRID:
  - functional replication BS (task farm, data parallel)
  - rule based autonomic managers
Sample autonomic manager: performance

- Master/worker implementation of embarrassingly parallel computations
- Performance model
- Policies
- Autonomic cycle applying policies:
  \[ \text{monitor} \Rightarrow \text{analyse} \Rightarrow \text{plan} \Rightarrow \text{execute} \]

\[ \max\{T_S, T_W/N_w, T_C\} \]

P1:: if \((T_s > \max\{\ldots\} \land T_s > \text{UserContract})\)
then addWorker

P2:: if \((T_s < \text{UserContract})\)
then removeWorker

P3:: ...
Sample autonomic manager: security

- Use/Provide component interaction
- Deployment info (with secure/insecure link taggings)
- Pre-defined proxy components (cypher, de-cypher)
- Policies
- Autonomic cycle applying policies:
  - monitor \(\Rightarrow\) analyse \(\Rightarrow\)
  - plan \(\Rightarrow\) execute
Sample run
Sample run
Sample run
This is:

- GCM
- Behavioural skeletons
- Business rule engine based autonomic manager
- Approximate performance models

\[ T_s = \max\{T_e, \frac{T_w}{n_w}, T_c\} \]
Appl. programmers:

- Provide “worker” components
- Instantiate a task farm behavioural skeleton
  - through proper ADL (XML) file

And that’s it!
Security management

A

Insecure network connection

B
Security management

- Information about deployment is *transparent to the user*
- Pipelined proxies (if possible)
Performance impact...
Performance impact...

Master/worker application: the more SSL connections you use, the higher the performance penalty.
Components vs. services ...

- services ≈ component ???
- mostly **YES**, but:
  - no info of those needed to *move* services
  - service = component - use ports
- as a consequence:
  - ideal to sell *static* applications
  - if the case, *dynamic* stuff is programmed *ad hoc*
Services are moving ...

- SCA (Service Component Architecture)
- composite components out of plain services
- composites are plain components
Services are moving ...

- SCA (Service Component Architecture)
- composite components out of plain services
- composites are plain components
GCM \( xx \) SCA

\( xx = \) vs. \( or \) \( xx = \) with?

- Hierarchical component composition
- Collective + data/stream ports
- Autonomic management of notable composite components
- XML based ADL
- Reference implementation in GridCOMP
  - Fractal based, in ProActive
GCM $xx$ SCA

$xx = \textbf{vs.} \ or \ xx = \textbf{with?}$

Already there, primitive

- Hierarchical component composition
- Collective + data/stream ports
- Autonomic management of notable composite components
- XML based ADL
- Reference implementation in GridCOMP
  - Fractal based, in ProActive
GCM $xx$ SCA

$xx = \text{vs. or} \ xx = \text{with?}$

Already there, primitive Can be implemented

GCM

- Hierarchical component composition
- Collective + data/stream ports
- Autonomic management of notable composite components
- XML based ADL
- Reference implementation in GridCOMP
  - Fractal based, in ProActive
GCM xx SCA

xx = vs. or xx = with?

Already there, primitive

Can be implemented

Experimented!

- Hierarchical component composition
- Collective + data/stream ports
- Autonomic management of notable composite components
- XML based ADL
- Reference implementation in GridCOMP
  - Fractal based, in ProActive
SCA task farm
GCM behavioural skeleton

- Behavioural skeleton in SCA:
  - JBoss rule based manager + task farm skeleton
  - preliminary experimental results: feasibility & scalability
  - JBoss rule based manager moved in GridCOMP GCM reference implementation (ProActive/Fractal)

- GCM task farm BS as a service:
  - generic, optimized, batch task processor (user defined tasks)
SCA GCM BS results

Autonomic reconfig (triggered by monitor events)

Half tasks computed in half time (2x workers)
Further, ongoing SCA/GCM activity

- providing collective *connector* components in SCA
  - collective out component (one to $N$, configurable distribution policy: scatter, {multi,broad,uni}cast)
  - collective in component ($N$ to one, configurable gathering policy: gather, reduce, combine)
- preliminary implementation ready, going to collect experimental results next month
Lessons learned

- Still poor support for dynamic component assemblies in SCA
  - 1.0 had something, looked like to improved in 1.1, almost disappeared in 1.2 ... (as perceived from the developer team)

- Service / component integration more or less perfect

- Support for a variety of host languages, bindings, etc.

- Open source project, with nice community behind, with recent official involvement of RedHat (pros&cons!)
Our “vision”

- Modules
- Objects
- Components
- Autonomic components
- Deployable, autonomic services
Our “vision”

- **Modules**
- **Objects**
- **Components**
- **Autonomic components**
  - Deployable, autonomic services

**Encapsulation**

**Time**

- **today**
Our vision (2)

❖ today:

algorithm complexity concerns
reusability issues
object code portability
...

Marco Danelutto Components for Grids: an evolutionary process EchoGrid meeting Athens, June 2008
Our vision (2)

- today:
  - Functional code/concerns
    - Perfectly working code, not so impressive global achievements
Our vision (2)

- today:
  - non functional concerns
  - Functional code/concerns
  
  Perfectly working code,
  fairly better global achievements
Our vision (2)

- today:

  - non functional concerns
  - application programmer concerns
  - perfectly working code,
    fairly better global achievements
Our vision (2)

- tomorrow:

  algorithms
  complexity concerns
  reusability issues
  object code portability
  ...

Our vision (2)

- tomorrow:

  Non functional concerns, application orchestration, autonomic management

Perfectly working application schema
Our vision (2)

❖ tomorrow:

- Non functional concerns, application orchestration, autonomic management
- Perfectly working application

functional co-processor
Our vision (2)

- tomorrow:

Non functional concerns, application orchestration, autonomic management

Perfectly working application
EchoGRID roadmap

- Already identified key “components” in this vision
- *Innovative* programming models in the roadmap fit the vision
- Autonomic management & services look like kind of *first-class citizens* in the roadmap ...
  - *and this must* be the focus