Project no. FP6-034442

GridCOMP

Grid programming with COMPonents : an advanced component platform for an effective invisible grid

STREP Project

Advanced Grid Technologies, Systems and Services

D.DIS.03 – Proceedings of the first GridCOMP workshop

Due date of deliverable: 30 November 2007

Actual submission date: 04 January 2008

Start date of project: 1 June 2006 **Duration**: 30 months

Organisation name of lead contractor for this deliverable: INRIA

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	PU

Keyword List: Grid, component, programming model, middleware, GCM
Responsible Partner: Denis Caromel, INRIA

MODIFICATION CONTROL			
Version	Date	Status	Modifications made by
0	DD-MM-YYYY	Template	Patricia HO-HUNE
1	20-11-2007	Draft	Cédric Dalmasso
2	03-01-2008	Draft	Cédric Dalmasso
2.1	04-01-2008	Draft	Yu Feng
2.2	14-01-2008	Draft	Cédric Dalmasso
2.3	15-01-2008	Final	Cédric Dalmasso

Deliverable manager

- Cédric Dalmasso, INRIA

List of Contributors

- Workshop presenters
- Denis Caromel, INRIA
- Cédric Dalmasso, INRIA

List of Evaluators

- Rajkumar Buyya, UoM
- Vladimir Getov, UoW

Summary

This document contains proceedings of the 1st GridCOMP workshop on Grid Component Model (GCM[1]) held in Beijing, on 31st October 2007, as one of the GRIDs@Work week events at the CNIC facilities, Beijing, China.

The main objective of this workshop was to show the current results produced by the GridCOMP project, including the explanation of the basic features provided by the GCM programming model, its implementation in ProActive, and also the benefits of using the GCM in several use cases, and finally some presentations of further upcoming perspectives.

The workshop was composed of three sessions, which were Basic Programming features, User presentations and Perspectives & Panel. For each presentation, we listed a brief description and slides in this document.

---

[1] The definition made in the CoreGRID NoE EU funded project is available at http://www.coregrid.net/mambo/images/stories/Deliverables/d.pm.04.pdf

Table of Content

1 Session 1: Basic Programming features

In this session, the presentations show the basic programming features provided by the GCM. After an overview of ProActive and GCM, the work done in the work packages 'Component framework implementation' and 'Non functional component features' is presented.

1.1 Presentations

ProActive and GCM: Status and future directions

Denis Caromel
UNSA, INRIA, IUF

This talk presents the current status of the GCM reference implementation in ProActive, and lists the last major improvements in the ProActive/GCM middleware such as API refactoring, upcoming implementation of the GCM Interoperability Deployment and GCM Application Description, etc.

Basic GCM Functionalities

Cédric Dalmasso
INRIA

This presentation aims to present concepts of the GCM programming model and how the concepts are used with the ProActive/GCM reference implementation. It shows the ability of the model and its implementation to provide solutions for parallel, distributed and multi-threaded computing.

The following principles are exposed:

- primitive and composite components,
- client server and non-functional (controller) interfaces with detailed presentation of multicast and gathercast cardinality.

Adaptative Behaviour with GCM

Marco Aldinuci and Nicola Tonellotto
University of Pisa, ISTI-CNR

In this presentation, we discuss our approach to autonomic behaviour with GCM. The key points of the GCM are the hierarchical compositional model, the advanced interactions among components and the management of the non-functional aspects of components. We focus on this last topic, exploiting the autonomic computing paradigm to control the QoS of GCM-based applications.
The key point of the autonomic computing paradigm is the autonomic feedback loop, in which a manager supervises a set of component and triggers corrective actions at run-time to reconfigure the components. This is done in order to satisfy user-level goals specified through QoS contracts. We introduce the concept of behavioural skeleton as a way to abstract parametric paradigms of component assembly, such as functional replication, proxy,

wrappers, etc. A behavioural skeleton is specialized to solve one or more management goals establishing a parametric orchestration schema of inner components.
As an example, we implemented the functional replication behavioural skeleton in the GCM, demonstrating the effectiveness of its autonomic management on a parameter sweep application (a Mandelbrot set generator). Then, we discussed some performance measures of its ProActive/Fractal implementation compared with the ASSIST implementation exploiting the same autonomic features.

2 Session 2: User presentations

The aim of this session is double:

1. Collect feedbacks from the user community, mainly GridCOMP industrial partner involved in the 'Use Case' work package (WP5).
2. Disseminate the first results achieved using GCM in real enterprise use cases, such as those developed in the GridCOMP project.

In addition to the presentation, some videos have been produced in order to show recording demo of the presented use cases.

2.1 Presentations and videos

Wrapping legacy PL/SQL enterprise code using GCM

Fabio Luiz Tumiatti
Atos Research and Innovation

The Use Case selected by Atos Origin uses PL/SQL-based source code, and the candidate application selected was the so called "Computing of DSO value". The DSO (Days Sales Outstanding) is the mean time that clients delay to pay an invoice to Atos. This information is needed by several internal departments as much updated as possible and the process lasts about 4 hours to compute around 6.600 clients.
The objective is to reduce the execution time without upgrading the infrastructure. With that it will be possible to update the information more frequently and maintain or reduce infrastructure cost. For that is necessary to avoid the rewriting of the PL/SQL code and make a good analyzes and distribution of the code between the master and nodes database to use with GCM.

Telecom Computing Application (include video)

Scientific Computing Application (include video)

Gastón Freire Amoedo
Grid Systems

These two presentations introduce the use case applications being analyzed by Grid Systems: Telecom Computing (EDR Processing) and Scientific Computing (Wing Design). Through the slides, the different aspects of the use cases are presented:

- Short summary of the current problems the applications are facing.
- High level block design of the solution to those problems, using GridCOMP.

- GridCOMP features leveraged by the use case applications.
- Benefits of the proposed solution.
- Architectural design of the prototypes.
- Short description of the main components and their relationships.

Finally, a video demonstrates the current prototypes in action: a test EDR file is generated and processed, and the performances of three wing configurations are simulated. These videos are available on the GridCOMP website[2].

Legacy code wrapping, interoperation with ChinaGrid Support Platform and Bioinformatics application

Weiyuan Huang
Tsinghua University

The aim of Legacy code wrapping is to develop techniques and methods for turning legacy code into components. This job includes two parts: extending ADL and defining the standard API for wrapping the legacy codes to components. At present, we have done most of the Java packages and have done a primary implementation of the design.

Interoperation with ChinaGrid Support Platform (CGSP) is an important job for GCM. In this part, we make GCM interoperate with CGSP through two ways. One is to wrap the core modules of CGSP as components and expose some interfaces for users to use. The other way is to treat CGSP as a deployment protocol. Users of ProActive/GCM could use it the same as Globus, SSH, etc.

Finally, a Bioinformatics application is shown. This application is composed of four independent parts. Each of the four parts is deployed as a component on an individual node, and each component runs the legacy code using legacy code wrapping.

Load-balancing/scheduling with multicast interfaces

Matthieu Morel
Universidad de Chile

In this presentation, improvements of the multicast interfaces are explained. The objective was to add the following new features:

- Automatic load-balancing (dynamic dispatch), in order to optimize the mapping of tasks to workers. Based on a simple but efficient (and potentially configurable) knowledge-based algorithm, that considers global relative processing speeds of workers (i.e. computational power + network latency). No prediction heuristics are used.
- automatic reduction of results
- unicast dispatch mode

These modifications provide a clear separation between:

---

[2] http://gridcomp.ercim.org/

1. partitioning of data
2. dispatch of invocations to workers
3. processing of results

They also provide a more configurable framework (partitioning, dispatch and reduction can all be customized).

Business Process Management Application (video)

Thomas Weigold
IBM

In this video, a demo of the IBM use case application, a biometric identification system, is showed. The core problem is to identify a given person solely on his biometric information by comparing its fingerprints against a large database of enrolled (known) identities. This requires massive computing power because biometric matching algorithms are non trivial and must be applied many times. Therefore, the identification system takes advantage of a Grid infrastructure and appropriate GridCOMP/GCM components, distributes the problem across the nodes, and this way achieves real-time identification performance.

Scheduling ProActive/GCM Applications on Global Grids using Gridbus Resource Broker

Xingchen Chu
The University of Melbourne

In this presentation, we present the design and implementation of seamlessly integrating two complex systems component-based distributed application framework ProActive/GCM and Gridbus Resource Broker.
The integration solution provides:

- the potential ability for component-based distributed applications developed using ProActive framework, which leverages the economy-based and data-intensive scheduling algorithms provided by the Gridbus Resource Broker;
- the execution runtime environment from ProActive for the Gridbus Resource Broker over component-based distributed applications.

We also present the evaluation of the integration solution based on examples provided by the ProActive/GCM distribution and some future directions of the current system.

Interoperability & Cooperation between ProActive and XServices

Yan Zhu
BUAA

The presentation focuses on the interoperability and cooperation between ProActive middleware from INRIA Sophia Antipolis OASIS Team and XServices Suite from Beihang University Web Services Team. First, it gave a short introduction of our Web Services team, including research fields, main achievements, and software productions. Secondly, SOA implementation architecture is represented with all the software of XServices Suite. Then, four main components inside XServices are to be discussed in detail, which are very important elements in Web Services Environment. Thirdly, some kernel application fields of

XServices Suite are shown, especially in the field of E-government, remote sensing, intelligent transportation system and collaborative seismic model. Finally, with the drive of OW2 Open Source issue and short Visit Scholar Programme of FP6 EchoGRID Project, the bridge between Proactive and XServices is built. It compares the differences and common ideas between them and introduces our approach to make them have the ability of interoperability and cooperation.

3 Session 3: Perspectives & Panel

To conclude the workshop, this session was dedicated to present further upcoming perspectives around GCM or in the GridCOMP project.

3.1 Presentations

Specifying GCM component with UML

Antonio Cansado and Eric Madelaine
UNSA, INRIA, CONICYT

The talk is in the frame of formal specifications of components. It gives a brief theoretical introduction on the specification of distributed components, in order to check for dynamic compatibility of components. Then, a prototype tool called Vercors Component Environment is presented as means to ease the development of component applications. This tool is based on UML2 profiles, having a two-fold approach: the architecture is specified with UML Component Diagram, and the behaviour of the components with UML State Machines. Be believed these diagrams are well known by engineers and allow us to hide the complexity of the underlying formalisms, while being suitable for exhaustive state-space verification (model-checking).

Finally, the presentation ends by proposing new extensions needed on the tool in order to deal with a broader set of GCM components. We expect these techniques to allow an effective use of COTS (Commercial off-the-shelf) components, by checking for compatibility flaws before deploying the application.

Component-based grid platforms and environments

Vladimir Getov and Artie Basukoski
University of Westminster

These slides give an overview of the Grid IDE (GIDE) tool developed in the GridCOMP project. The GIDE composition perspective provides a toolbox with a list of components and with a set of tools so that applications can be visually composed using components. Although it functions similar to a drawing package, the back-end of the composition perspective generates necessary GCM specific development artefacts such as component definition ADL files and Java interface definitions. Figure 1 shows a general screen layout of the composition perspective. Figure 2 shows part of the domain model description that is used to implement the GMF backend which we use for the composition view. Finally, figure 3 shows a block diagram indicating the interactions of the GMF backend (via the diagram and composition structures) with the composition view.

The composition perspective also supports importing of ADL files and exporting of compositions to ADL files.

4 Conclusion

The presentation of the basics GCM features, code composition and autonomic management of GCM application, and additionally the use case presentations have showed the benefits that software developer and architect have to use and leverage the GCM programming model.

Also, some user presentations exposed improvements and the possible integration and cooperation with other framework and middleware. And finally, the presentation of attractive perspectives with the GIDE and model checking tool concluded the workshop.

Overall sessions, this first workshop on the GCM allowed the presentation of the first result produced in the GridCOMP project and disseminate the GCM.

5 Annex: Presentations

Session 1 - Basic Programming features

GridCOMP Workshop
October 31st, 2007
CNIC, Beijing, China

ProActive and GCM: Status and future directions

Denis Caromel

GridCOMP Scientific Coordinator

Denis.Caromel@inria.fr

Beijing, October 2007

GCM Partners

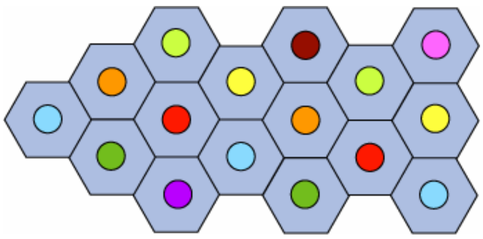
- GCM: Grid Component Model
  - GCM Being defined in the NoE CoreGRID (42 institutions)
  - Open Source ObjectWeb *ProActive* implements a preliminary version of GCM
  - Service Oriented: NESSI relation

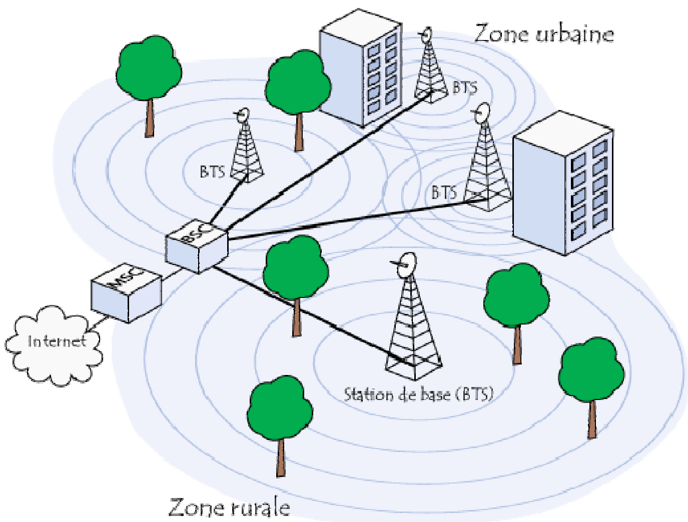
The vision: GCM to be the GRID GSM

- GridCOMP takes:
  - GCM as a first specification,
  - *ProActive* as a starting point, and Open Source reference implementation.

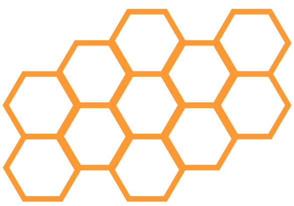
GSM and GCM Pictures

- GSM cells:





- Components:



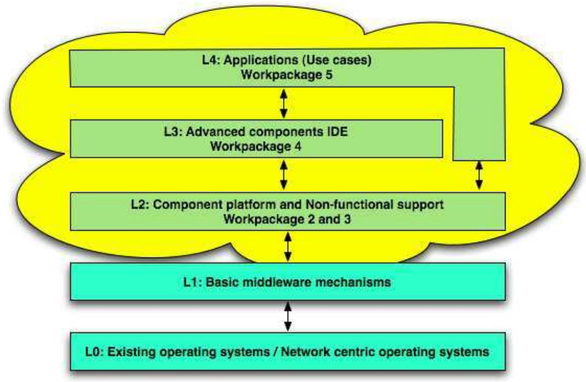
GSM and GCM Pictures

GSM and GCM Pictures

Overview of Project

Objectives:

GRID PROGRAMMING WITH COMPONENTS: AN ADVANCED COMPONENT PLATFORM FOR AN EFFECTIVE INVISIBLE GRID



- Interoperability Objectives:
  - Interoperability with other standards: EGEE gLite, UNICORE, NorduGrid, Globus, Web Services, LSF, IBM LL, SGE, etc.,
  - **A GCM ETSI Official Public Standard**

ETSI GCM TC Grid Standard

Work Item No 1

- GCM Interoperability Deployment
- GCM Application Description
- GCM Fractal ADL
  (Architecture Description Language)
- GCM Management (Java, C, WSDL API)

Form of GCM Interoperability Deployment

- Just an XML Schema:

  Specifies the deployment of the application

  Virtual Nodes

  onto the infrastructure (machine, OS, protocols, schedulers, etc.)

- Example: EGEE gLite schema:

```
<paext:gLiteGroup id="gliteGroup" virtualOrganisation="proactive" JDLFileName="job.jdl" jobType="normal"
    retryCount="3">
    <paext:requirements>
        other.GlueCEUniqueID == "grid10.lal.in2p3.fr:2119/jobmanager-pbs-sdj"
    </paext:requirements>
    <paext:rank>-other.GlueCEStateEstimatedResponseTime</paext:rank>
    <paext:stdout>stdout.log</paext:stdout>
    <paext:stderr>error.log</paext:stderr>
```


GCM Technical Structure

1. **Component Specification as an XML schema or DTD**
2. **Run-Time API defined in several languages**
   C, Java
3. **Description and Information for Deployment**
   (XML DD, Virtual Nodes, File Transfer, …)
3. **Packaging described as an XML schema**

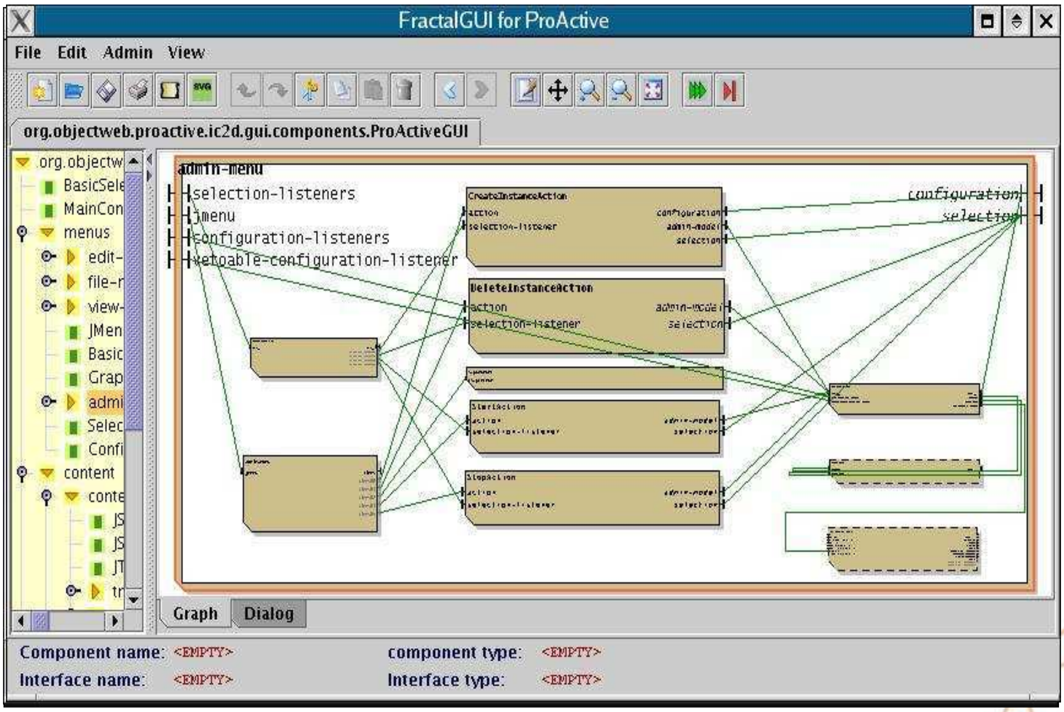
Status of GCM in ProActive

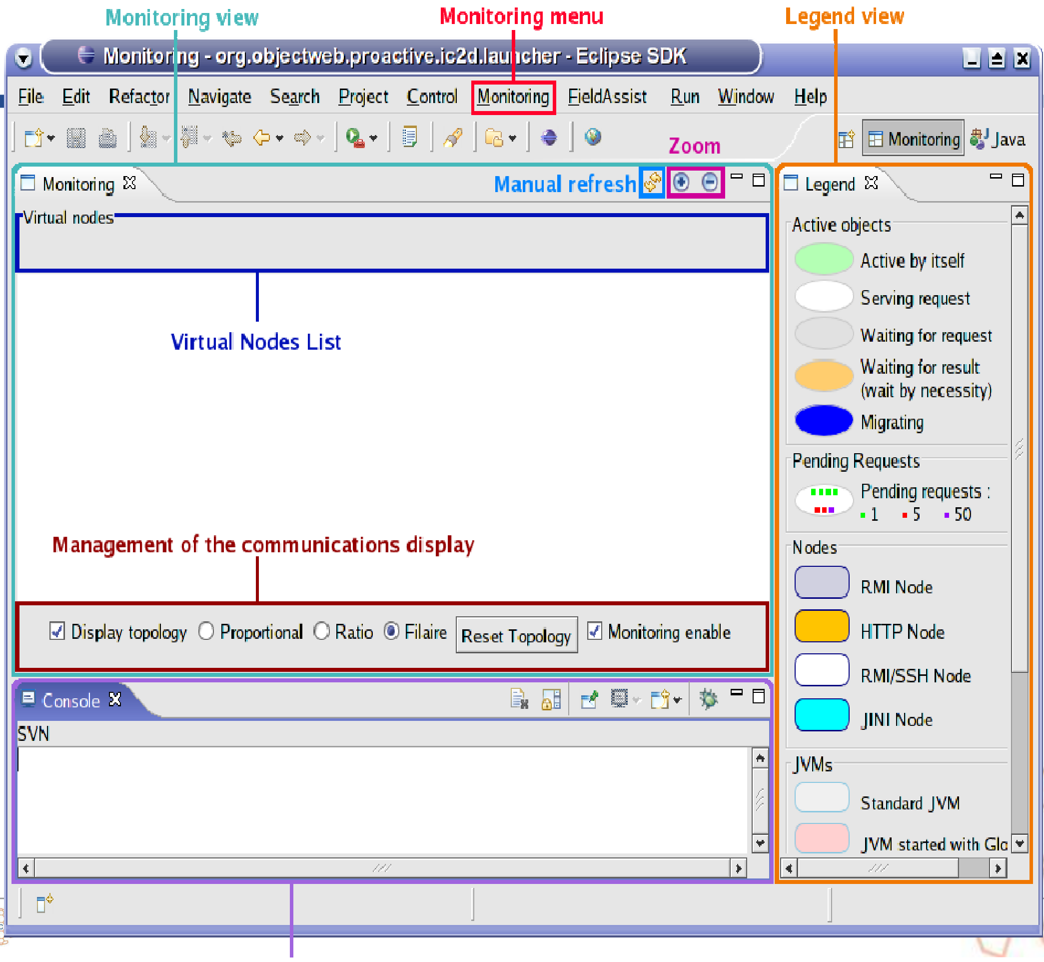
- Partial implementation:
  - ADL schema, API, Multicast, Gathercast, …
  - Component GUI (prototype)
- Distributed components for various applications:
  - Numerical, Legacy, …
- Achieved experiments:
  - A component application on up to 300+ CPUs

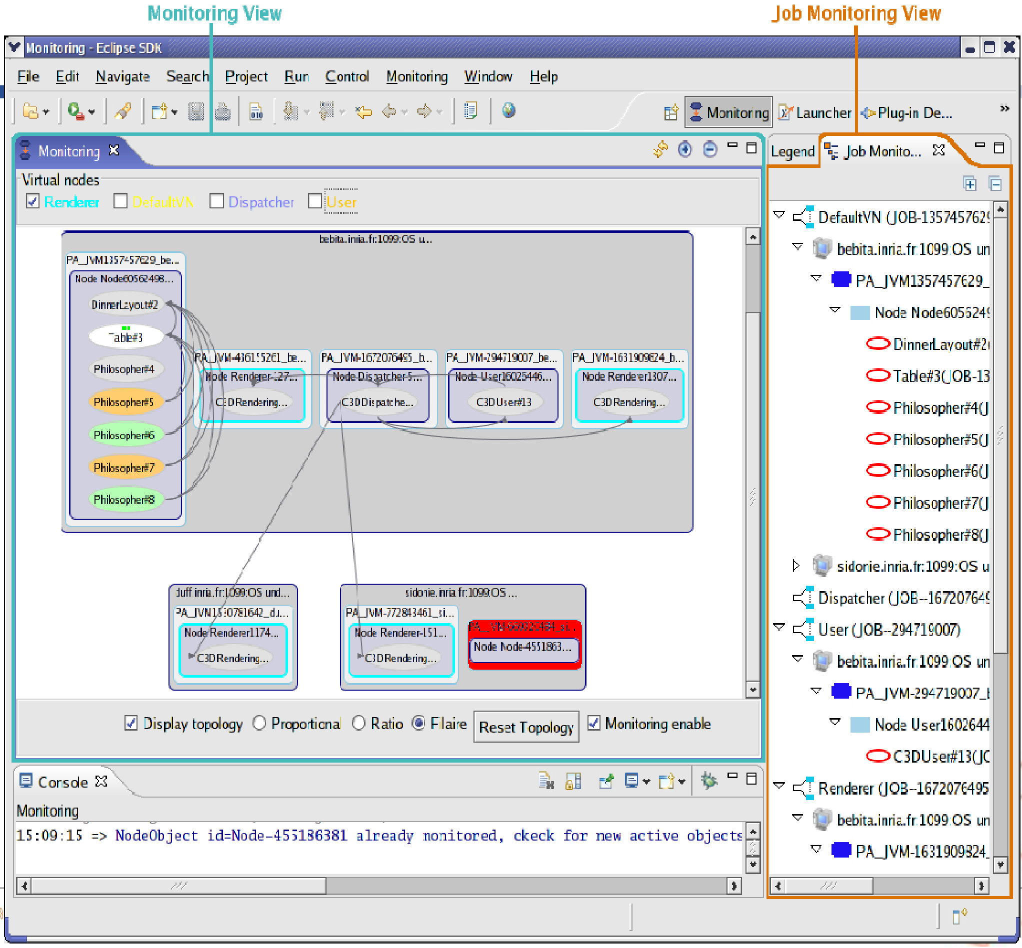
IC2D and Generic Environment

Eclipse GUI

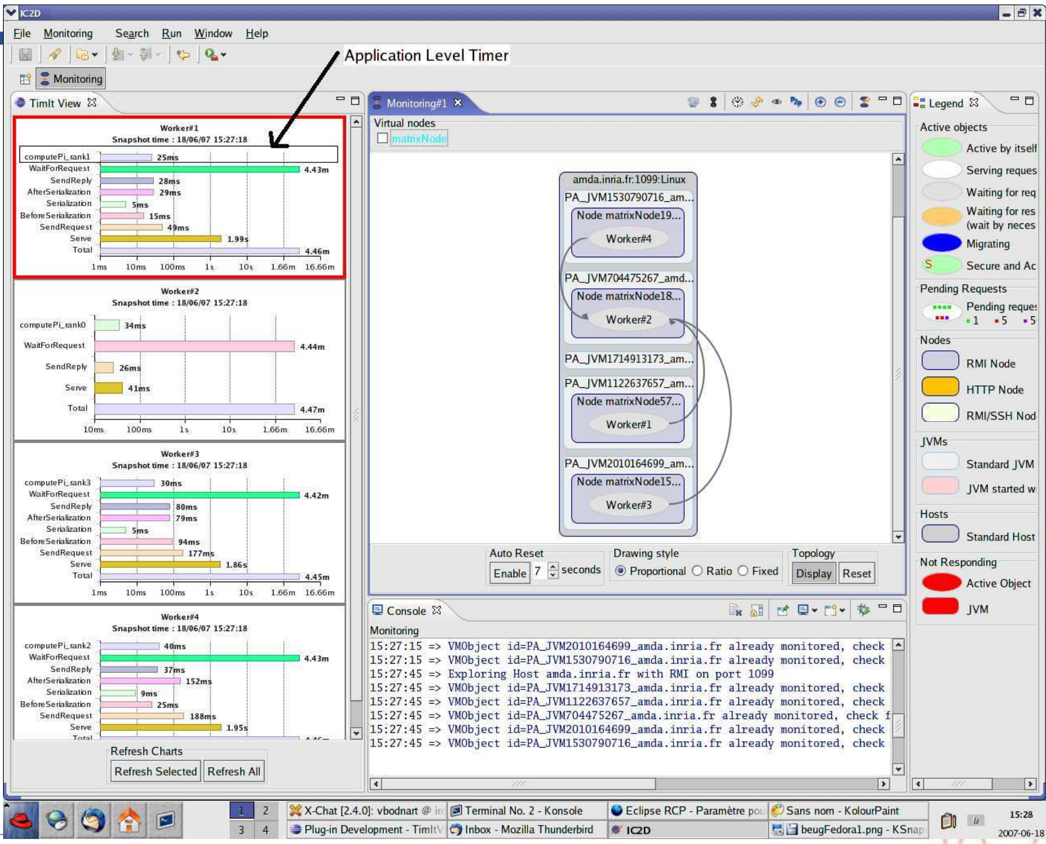
Prototype : GUI for Components



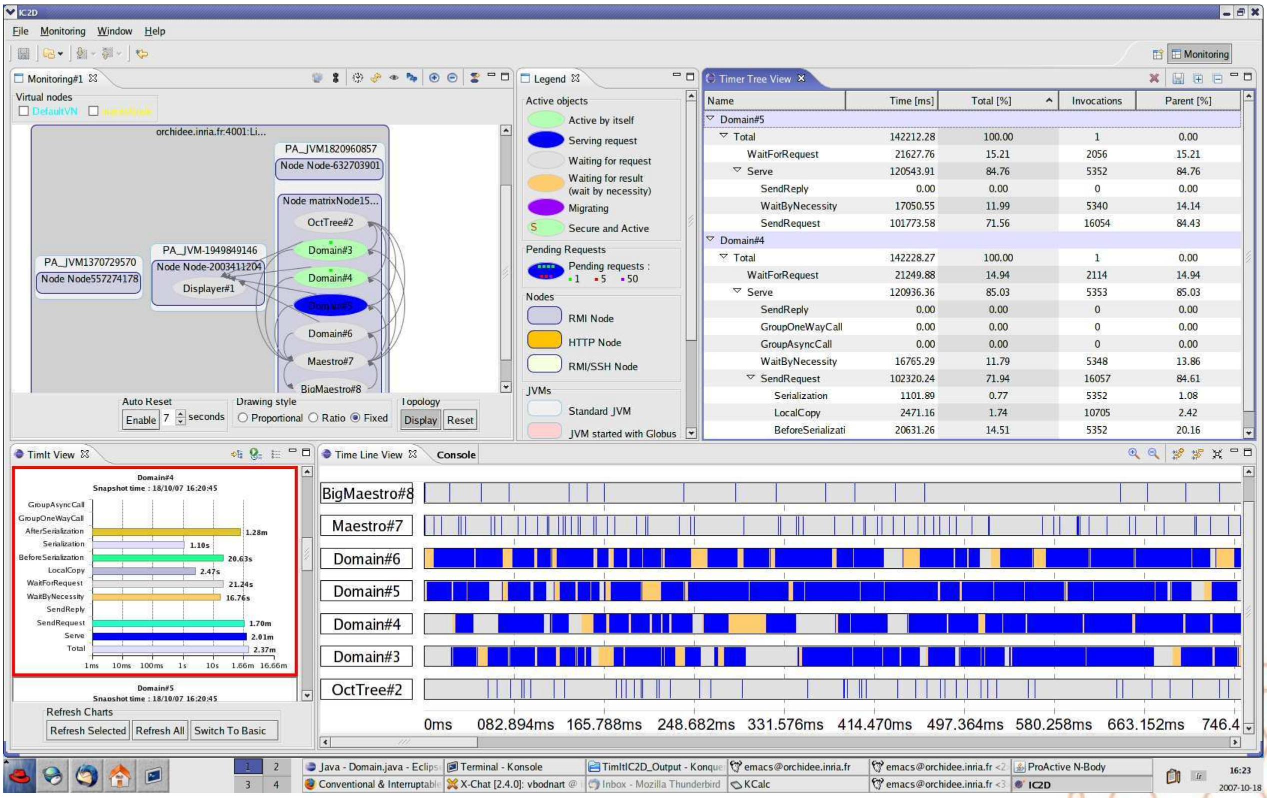




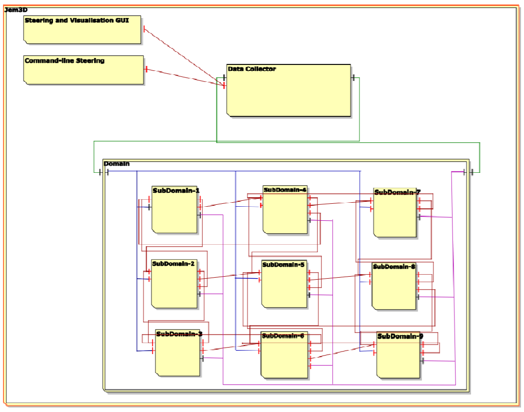
TimIt Automatic Timers in IC2D



ProActive / GCM Environment

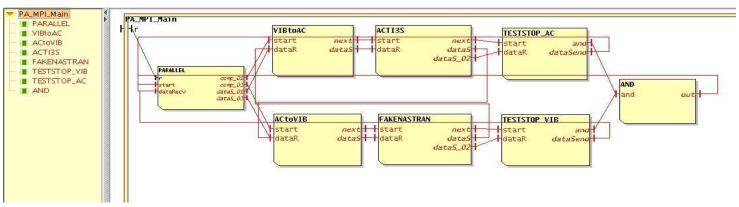


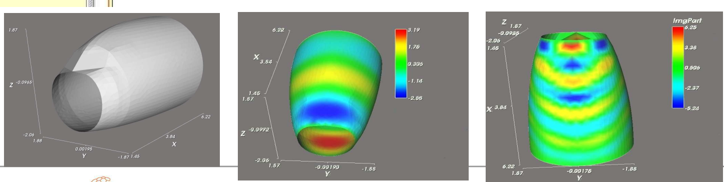
Current GCM experiments in ProActive



- JEM3D: 3D electromagnetic application:

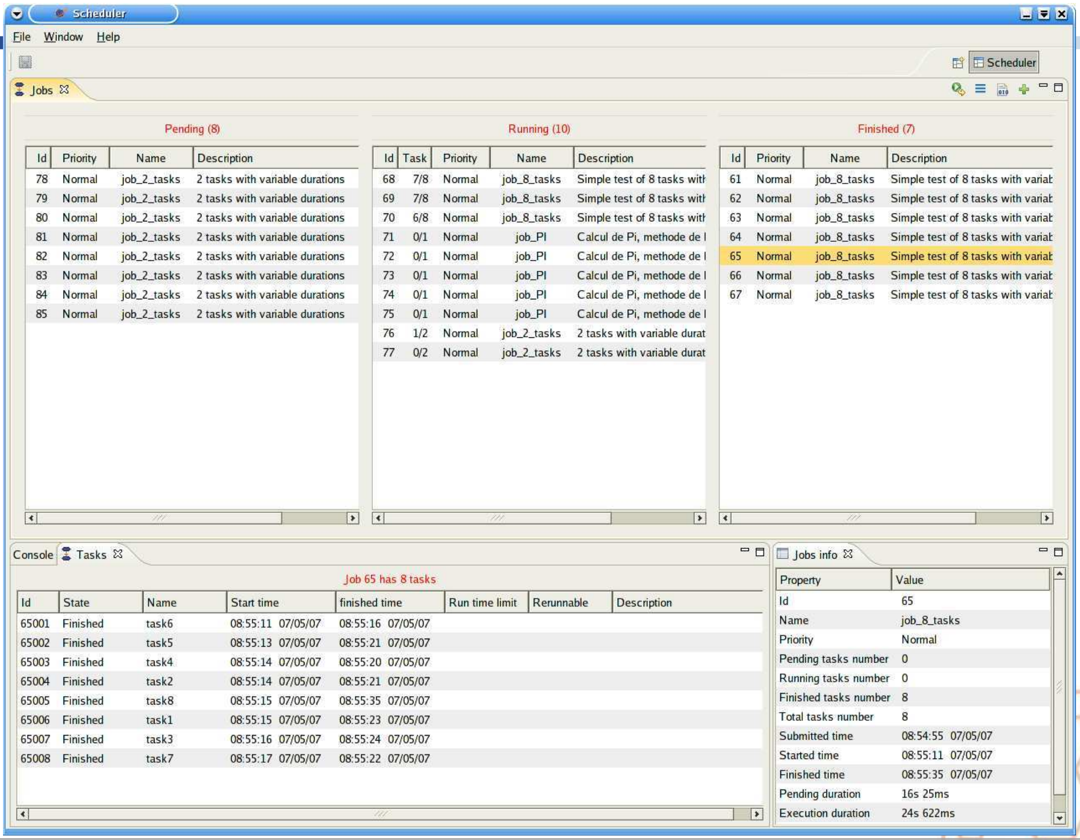
  a single Cp on 300+ CPUs on Grid



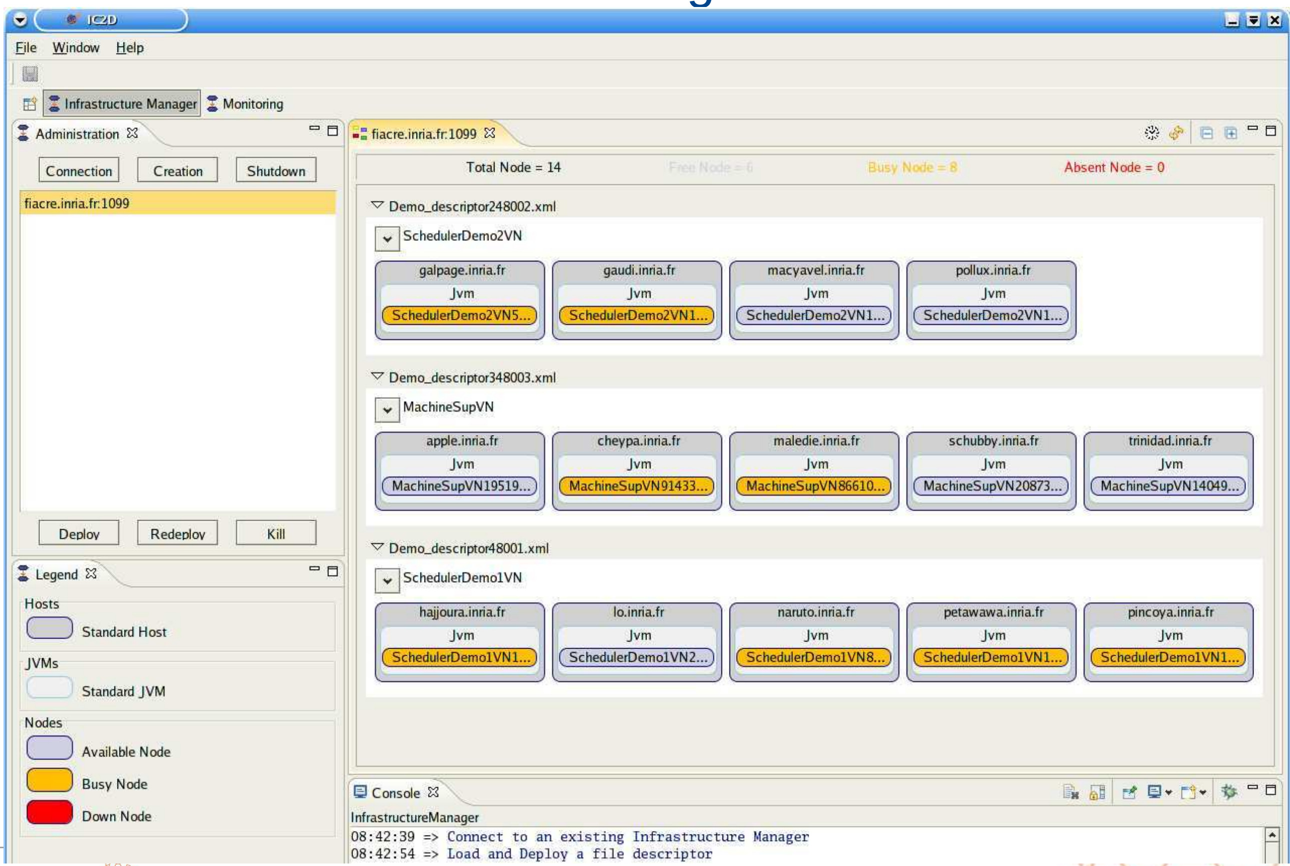


- Vibro-Acoustic application with EADS (legacy MPI coupling)

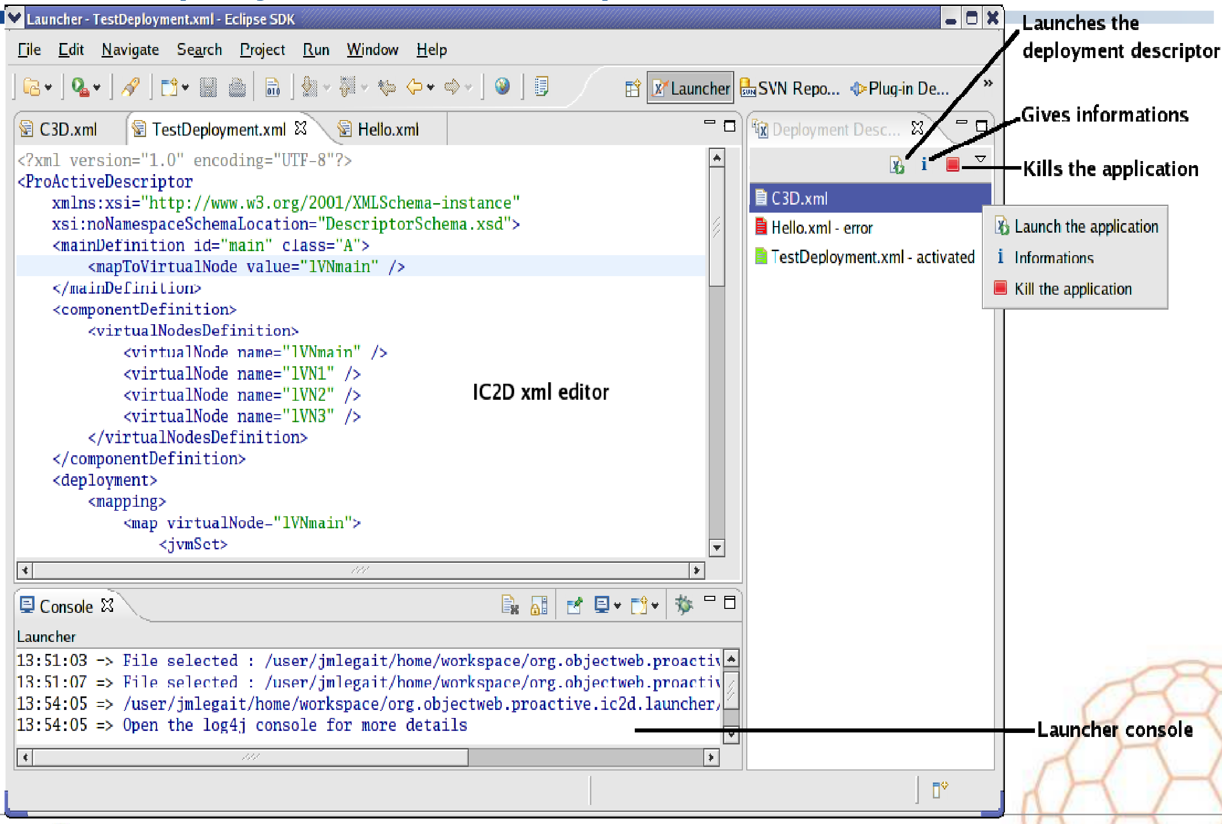
Scheduler: User Interface



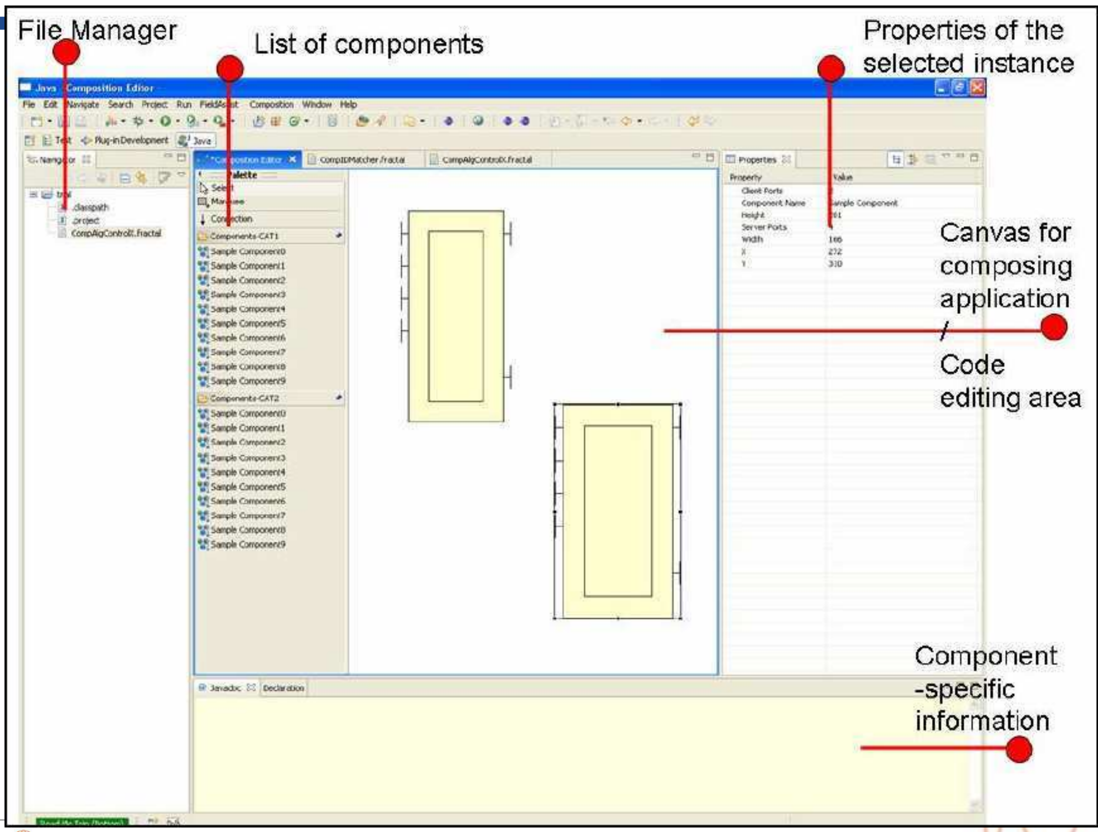
Scheduler: Resource Manager Interface



XML Deployment Descriptors



Component GUI under Dev. at Westminster Univ.



Grid programming with components:
an advanced *COMP*onent platform
for an effective invisible grid

Basic GCM Functionalities

Cédric Dalmasso, Antonio Cansado

INRIA - OASIS Team

Beijing, October 2007

GCM Components

- GCM: Grid Component Model
  - GCM was defined in the NoE CoreGRID
  - GCM extends Fractal with Grid specificities
- Open Source ObjectWeb *ProActive*
  - implements a preliminary version of GCM
- GridCOMP takes:
  - GCM as a first specification,
  - *ProActive* as a starting point, and
    Open Source reference implementation.

Scopes and Objectives:
Grid Codes to Compose and Deploy
No programming, No Scripting, …

Introduction to Components

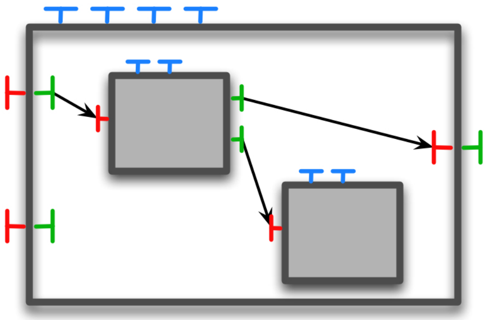
- What are software components?
  - Modules exposing the interaction with the environment
    - Provided (server) interfaces
    - Required (client) interfaces
  - Black-boxes (from outside)
- Advantages
  - Encapsulation (black-boxes)
  - Composition
  - Standardized Description ⇨ ADL ⇨ GUI, Verification
  - Units of deployment
  - Programming in the large vs. programming in the small (objects)
- Goal
  - Reuse and compose
  - Commercial Off-The-Shelf (COTS)

Rationale: Grid applications

Requirements	**Solutions with ProActive/GCM**
Distribution	Distributed components
Multiple administrative domains	Handled by the middleware
Heterogeneity	Portable implementations, interoperability
Legacy code	Encapsulation, interoperability
Performance	Legacy code, parallelism
Complexity	Hierarchies, collective interfaces
Dynamicity	Adaptation and coherent reconfigurations
Tools	ADL, GUI, Packaging

Approach Based on the Fractal Model

- INRIA - France Telecom, V1 in ’02
  - General model, core concepts
    - – Encapsulation
    - – Strict Definition
    - – Assembly and deployment units
  - Simple, extensible, hierarchical, dynamic
  - Separation of concerns (controllers)
  - However:
    - – Distribution ?
    - – Deployment ?
    - – Parallelism ?



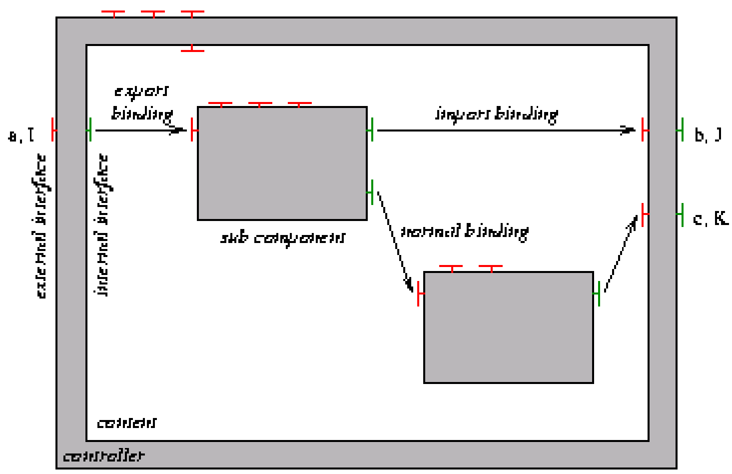
⇨ Fractal requires extensions for Grid Computing

⇨ Specified in the Grid Component Model - GCM (CoreGRID)

Some important Fractal Concepts

- Content
- Controller (or membrane)
- Server Interface
- Client Interface

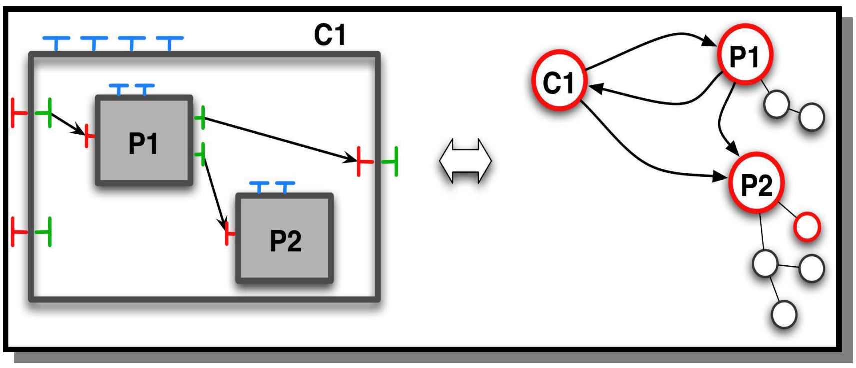
- Bind(ing)
- Functional interface
- Control (or non-functional) Interface



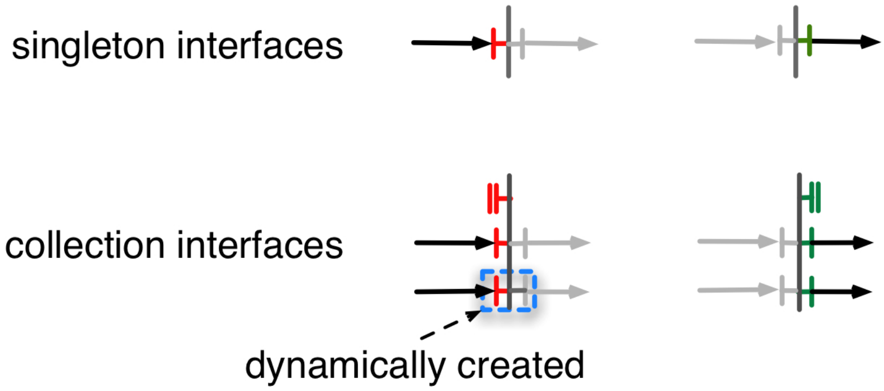
ProActive/Fractal

- Implementation of Fractal based on ProActive middleware Model
  - Based on MOP architecture: Component as Active Object
  - Distributed components, asynchronous communications (futures)
  - Benefits from underlying features of the middleware
    - Middleware services (Fault Tolerance, Security, Mobility etc..)
    - Deployment framework (in development GCM deployment, being standardized at ETSI)
  - Sequential processing of requests in each component
  - Main extensions to Fractal: deployment, collective interfaces
  - Configurable and extensible

ProActive/Fractal



Standard Fractal Interfaces



Only 1 to 1 communications!

GCM Collective Interfaces

- ⇨ collective interfaces
  - Multicast
  - Gathercast
    gather-multicast
- Simplify the design and configuration of component systems
- Expose the collective nature of interfaces
- Interface typing ➔ Verifications

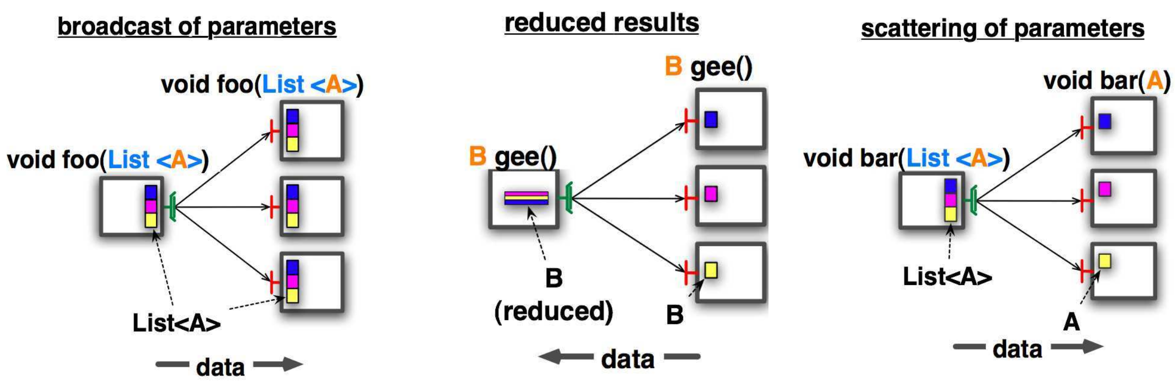
➔ **The framework handles collective behaviour at the level of the interface**

GCM Multicast interfaces

single invocation ⇨ list of invocations

- Multiple invocations
  - Parallel
  - Asynchronous
  - Selective
  - Dynamic
- Data distribution
  - Automatic
  - Customized distribution function
    - Broadcast, scattering, reduction
  - Explicit typing,
    - Parameterized collections
    - Compatibility verified at runtime when binding

Multicast Interfaces Illustrated



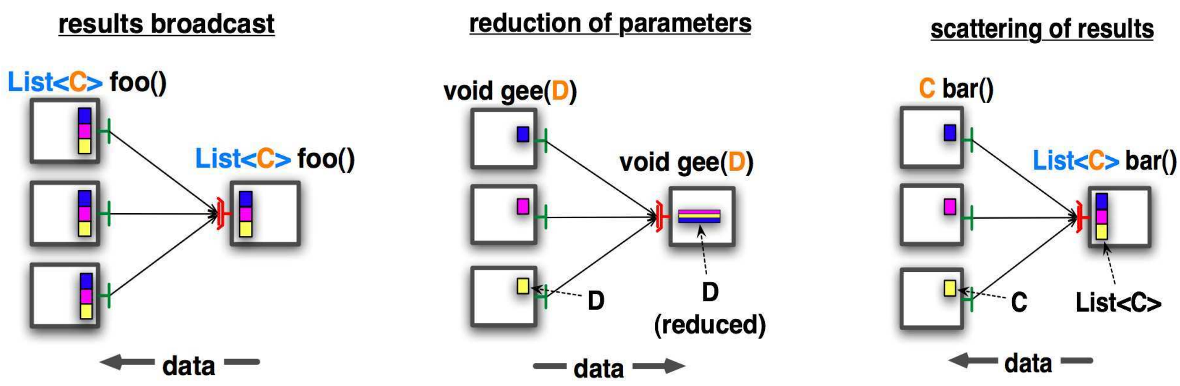
Configurable distribution policies
Parallelism
Strong typing

GCM Gathercast Interfaces

list of invocations ⇨ single invocation

- Synchronization
  - ~ “join” invocations
  - Customizable: wait-for-all, wait-for-some
  - Timeout
- Data distribution
  - Aggregation / reduction of parameters
  - Redistribution of results
  - Symmetrical to multicast

Gathercast Interfaces Illustrated



Configurable distribution policies
Synchronization
Strong typing

Architecture Description Language (ADL)

- Specifies the system architecture
  - Components, subcomponents
  - Bindings
  - Interfaces (IDL)
- Used to configure and deploy component systems

Architecture Description Language (ADL)

- In GCM, the Fractal ADL has been extended:
  - allows to reuse ProActive-specific features like deployment
  - supports Collective Interfaces

Virtual Nodes

```
<virtualNodesDefinition>
  <virtualNode name="Dispatcher" property="unique_singleAO"
      />
  <virtualNode name="Renderer" property="Multiple"
               constraintFile="RendererConstraints.xml" />
</virtualNodesDefinition>
```

- Permits a program to generate automatically a deployment plan:
  - find the appropriate nodes on which processes should be launched.

Virtual Nodes in the ADL

```
<exportedVirtualNodes>
  <exportedVirtualNode name="VN1">
    <composedFrom>
      <composingVirtualNode component="this" name="myNode"/>
    </composedFrom>
  </exportedVirtualNode>
</exportedVirtualNodes>
...
<virtual-node name="myNode" cardinality="single"/>
```

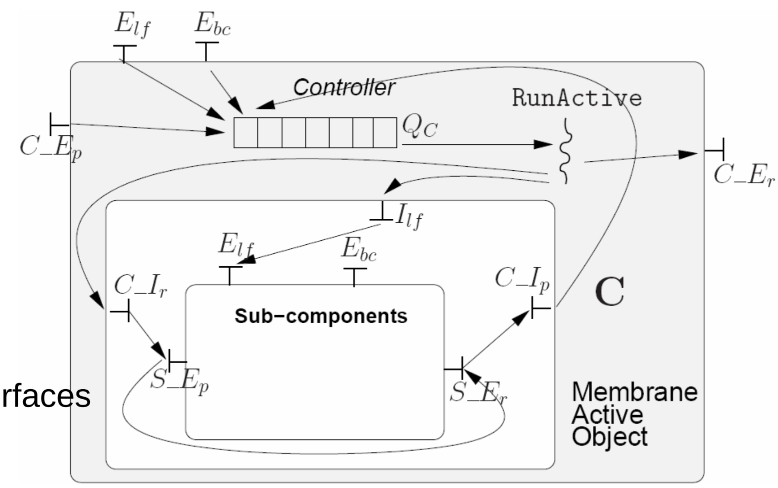
- Renames a VN
- Exports a VN name

➔ final version of the GCM specification will precisely define the syntax for the virtual node definition, and their composition.

http://proactive.objectweb.org

Let's practice a little more !

First-steps in GCM/ProActive Components



- Composite
  - Defined in ADL
- Primitive
  - Defined in ADL
  - Java class
    - implements server interfaces
- Interfaces
  - Cardinality (single or multiple) → ADL
  - Signed by Java interfaces
    - Distribution policy → Java annotations

Distribution Policy

- Given by Java annotations

```
@ClassDispatchMetadata(
  mode=@ParamDispatchMetadata(
         mode=ParamDispatchMode.BROADCAST))

interface MyMulticastItf {
        public void foo(List<T> parameters);
}
```

Grid programming with components:
an advanced COMPonent platform
for an effective invisible grid

GridCOMP
Effective Components for the Grids

Adaptative Behavior with GCM

Marco Aldinucci, M. Danelutto, S. Campa
University of Pisa, Italy

D. Laforenza, N. Tonellotto, P.Dazzi
ISTI-CNR, Italy

October 31th, 2007
Beijing, China
北京 - 中华人民共和国

Outline

- Motivation
  - why adaptive and autonomic management
  - why skeletons
- Behavioural Skeletons
  - parametric composite component with management
  - functional and non-functional description
  - families of behavioural skeletons
- GCM implementation
  - some hints today, much more tomorrow
    - Nicola Tonellotto and Patrizio Dazzi talk at "ProActive and GCM Tutorial and Hands-On Grid Programming" (Thursday 15,30-16,30)
  - preliminary experiments and performances

CGM model key points

- Hierarchic model
  - Expressiveness
  - Structured composition
- Interactions among components
  - Collective/group
  - Configurable/programmable
  - Not only RPC, but also stream/event
- NF aspects and QoS control
  - Autonomic computing paradigm

Why Autonomic Computing

- // programming & the grid
  - concurrency exploitation, concurrent activities set up, mapping/scheduling, communication/synchronisation handling and data allocation, ...
  - manage resources heterogeneity and unreliability, networks latency and bandwidth unsteadiness, resources topology and availability changes,

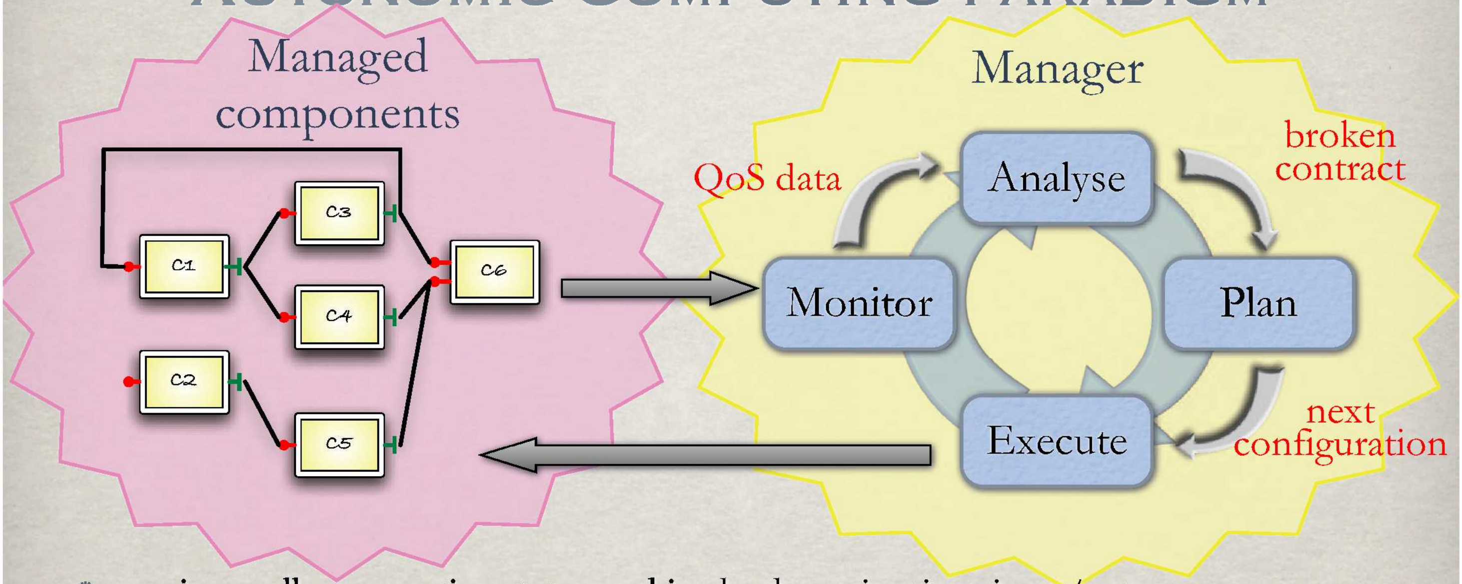
... and a non trivial QoS for **applications**

not easy leveraging only on middleware

our approach:

high-level methodologies + tools

AUTONOMIC COMPUTING PARADIGM



- monitor: collect execution stats: machine load, service time, input/output queues lengths, ...
- analyse: instantiate performance models with monitored data, detect broken contract, in and in the case try to detect the cause of the problem
- plan: select a (predefined or user defined) strategy to re-convey the contract to validity. The strategy is actually a "program" using execute API
- execute: leverage on mechanism to apply the plan

WHY SKELETONS 1/2

- Management is difficult
  - Application change along time (ADL not enough)
  - How "describe" functional, non-functional features and their inter-relations?
  - The low-level programming of component and its management is simply too complex
- Component reuse is already a problem
  - Specialising component yet more with management strategy would just worsen the problem
  - Especially if the component should be reverse engineered to be used (its behaviour may change along the run)

Why skeletons 2/2

- Skeletons represent patterns of parallel computations (expressed in GCM as graphs of components)
- Exploit the inherent skeleton semantics
  - thus, restrict the general case of skeleton assembly
  - graph of any component ➠ parametric networks of components exhibiting a given property
  - enough general to enable reuse
  - enough restricted to predetermine management strategies
- Can be enforced with additional requirements
  - E.g.: Any adaptation does not change the functional semantics

Behavioural Skeletons idea

- Represent an evolution of the algorithmic skeleton concept for component management
  - abstract parametric paradigms of component assembly
  - specialized to solve one or more management goals
    - self-configuration/optimization/healing/protection.
- Are higher-order components
- Are not exclusive
  - can be composed with non-skeletal assemblies via standard components connectors
    - overcome a classic limitation of skeletal systems

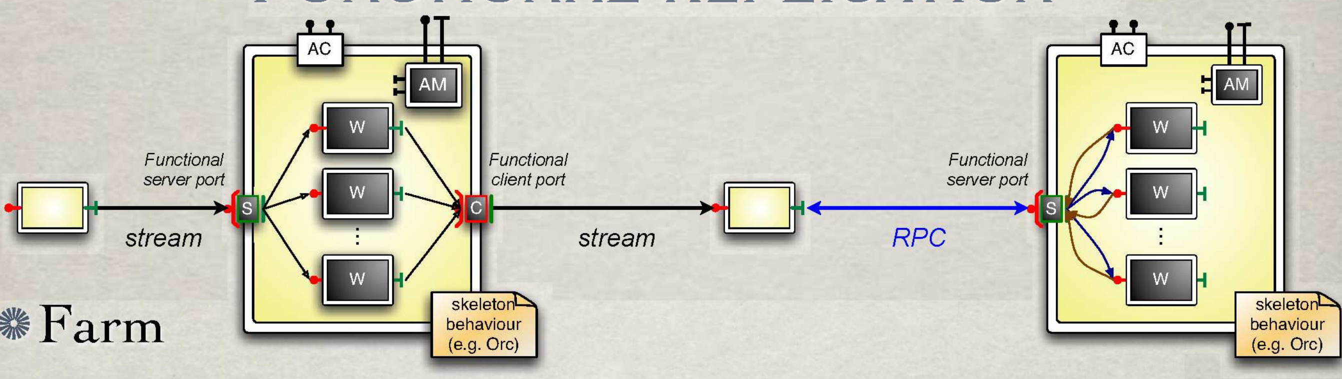
Behavioural Skeletons proprieties

- Expose a description of its functional behaviour
- Establish a parametric orchestration schema of inner components
- May carry constraints that inner components are required to comply with
- May carry a number of pre-defined plans aiming to cope with a given self-management goal
- Carry an implementation (they are factories)

Be-Skeletons families

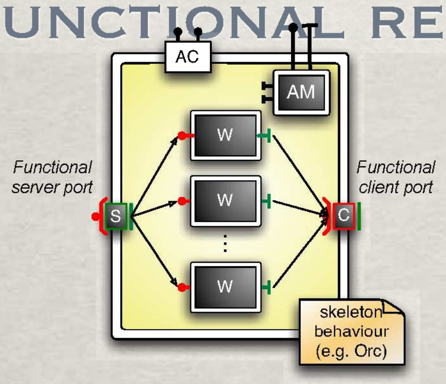
- Functional Replication
  - Farm/parameter sweep (self-optimization)
  - Simple Data-Parallel (self-configuring map-reduce)
  - Active/Passive Replication (self-healing)
- Proxy
  - Pipeline (coupled self-protecting proxies)
- Wrappers
  - Facade (self-protection)
- Many others can be borrowed from Design Patterns

Functional replication



- Farm
  - S = unicast, C = from_any, W = stateless inner component
- Data Parallel
  - S = scatter, C = gather, W = stateless inner component
- Fault-tolerant Active Replication
  - S = broadcast, C = get_one_in_a_set, W= stateless inner ...
- ...

Functional replication

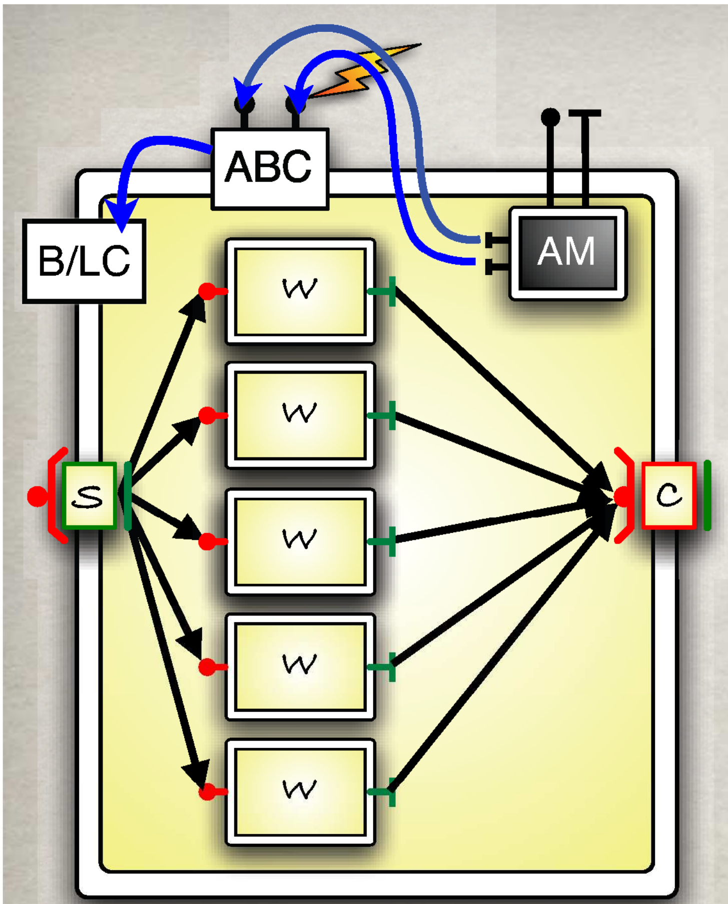


Functional behaviour description (orchestration)

$$system(data, S, G, W, in, out, N) \triangleq$$
$$S(data, in) \mid (| \; i : 1 \leq i \leq N : W_i(in_i, out_i)) \mid C(out)$$
$$W_i(in_i, out_i) \triangleq$$
$$in_i.get > tk > process(tk) > r > (out_i.put(r) \mid W_i(in_i, out_i))$$

- Meant to parametrically expose all allowed adaptation
  - Any AM policy that does not change this semantics is *correct*
  - As an example changing ***i*** in this schema is correct
  - Functional semantics is invariant from ***i***, non-functional one is not (and changing ***i*** means changing the number of Ws for self-* purposes

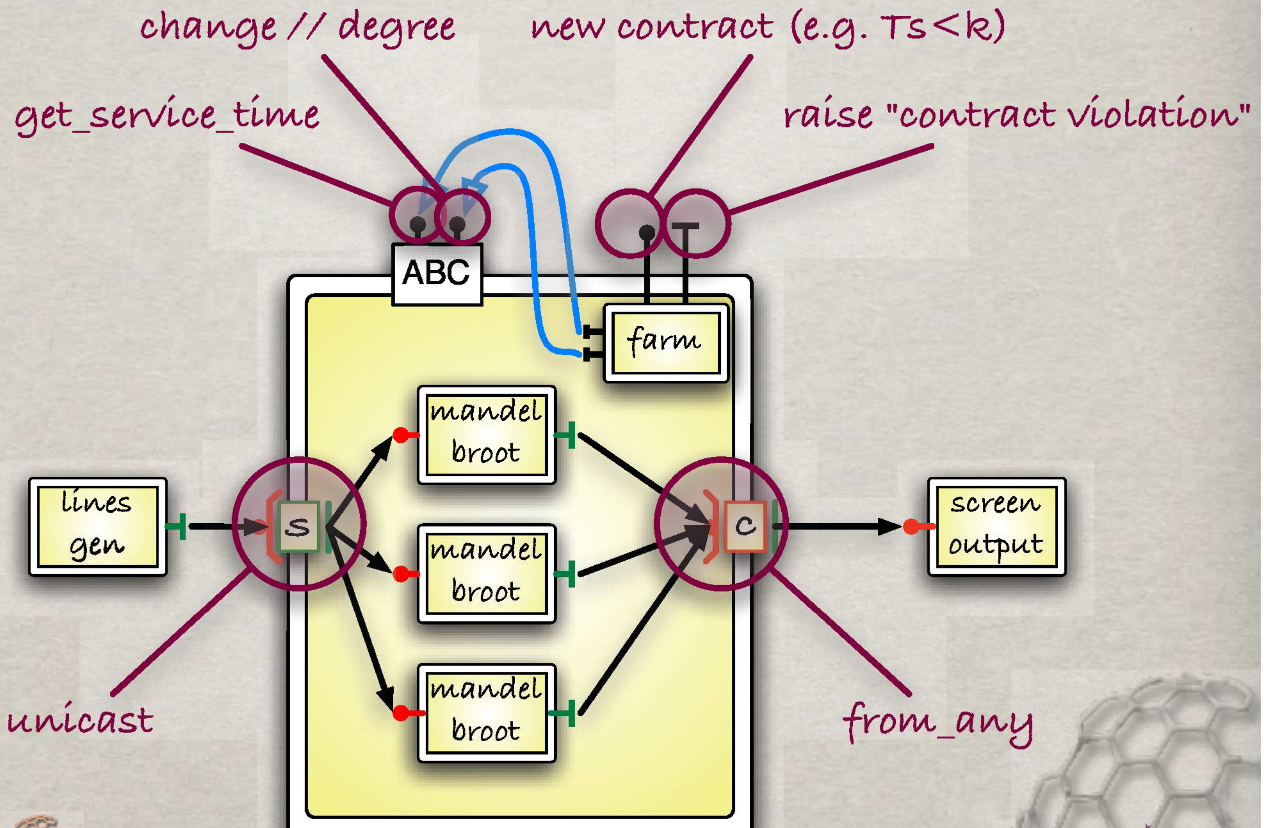
GCM Implementation



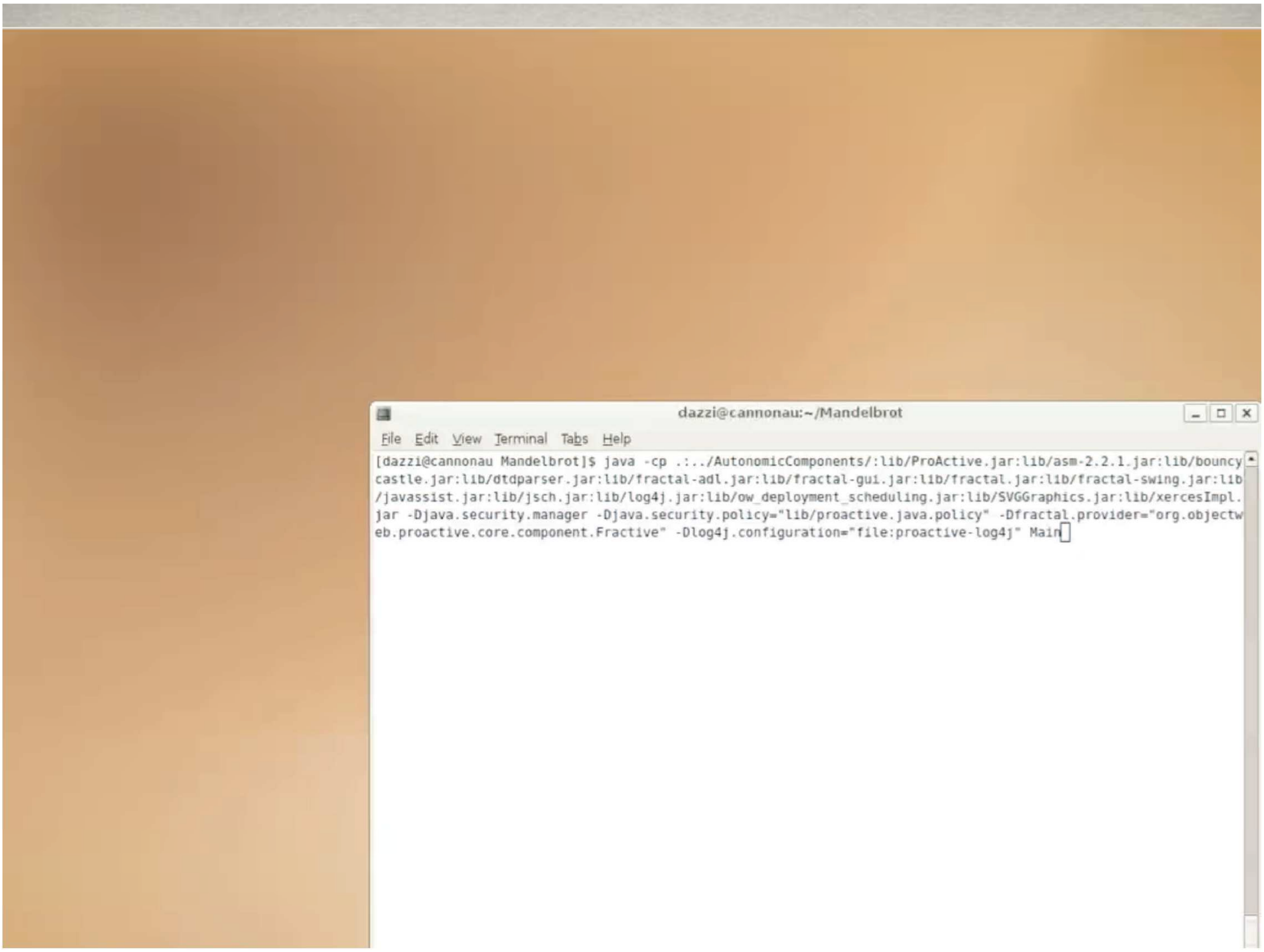
ABC = Autonomic Behaviour Controller (implements mechanisms)
AM = Autonomic Manager (implements policies)
B/LC = Binding + Lifecycle Controller

1. Choose a schema (e.g. functional replication) ABC API is chosen accordingly
2. Choose an inner component (compliant to Be-Ske constraints)
3. Choose behavior of ports (e.g. unicast/from_any, scatter/gather)
4. Wire it in your application. Run it, then trigger adaptations
5. Possibly, automatize the process with a Manager

Farm example (Mandelbroot)



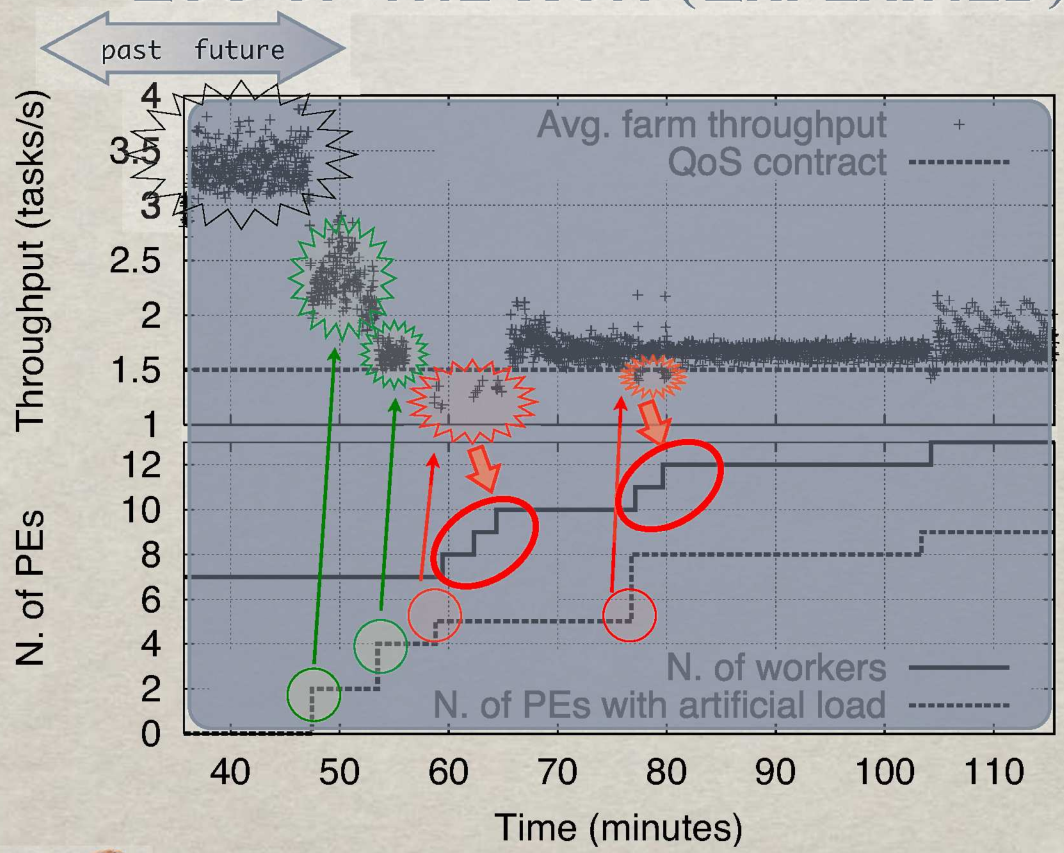
Grid programming with components: an advanced COMPonent platform for an effective invisible grid
CoreGRID: The European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and P2P Technologies



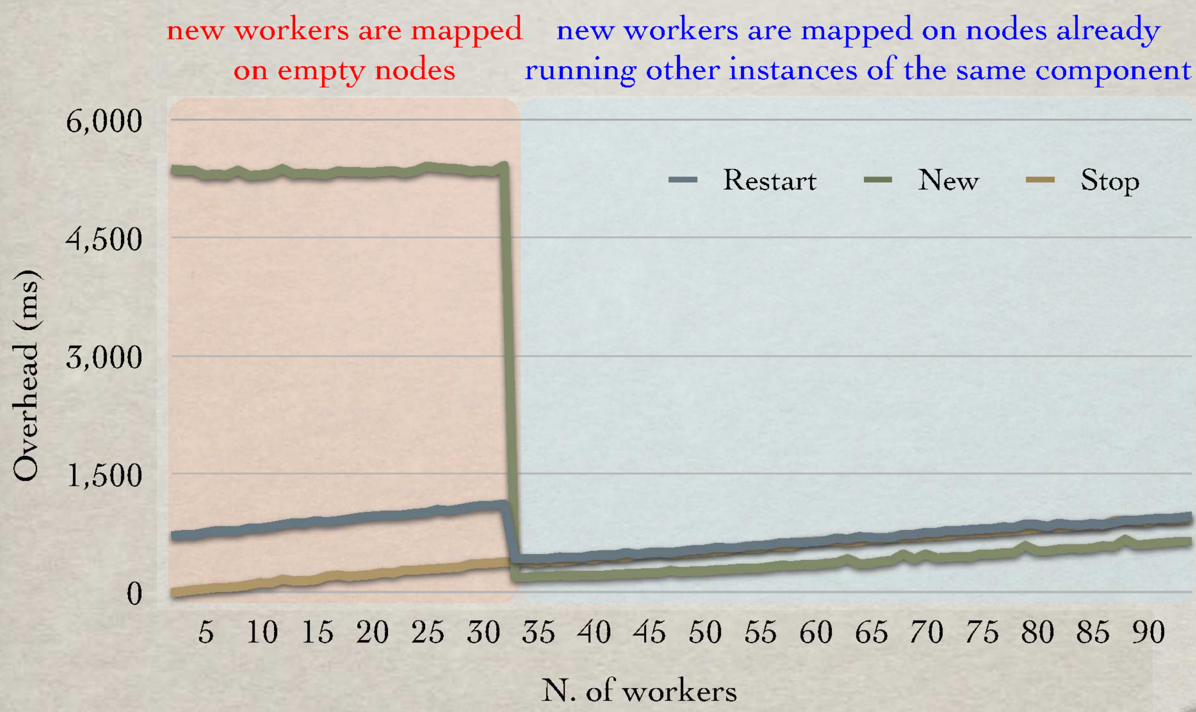
Not just farm (i.e. param sweep)

- Many other skeletons already developed for GCM
  - some mentioned before
- Easy extendible to stateful variants
  - imposing inner component expose NF ports for state access
- Policies not discussed here
  - expressed with a `when-event-if-cond-then-action` list of rules
  - some exist, work ongoing ...

Log of the Run (Explained)



Overheads



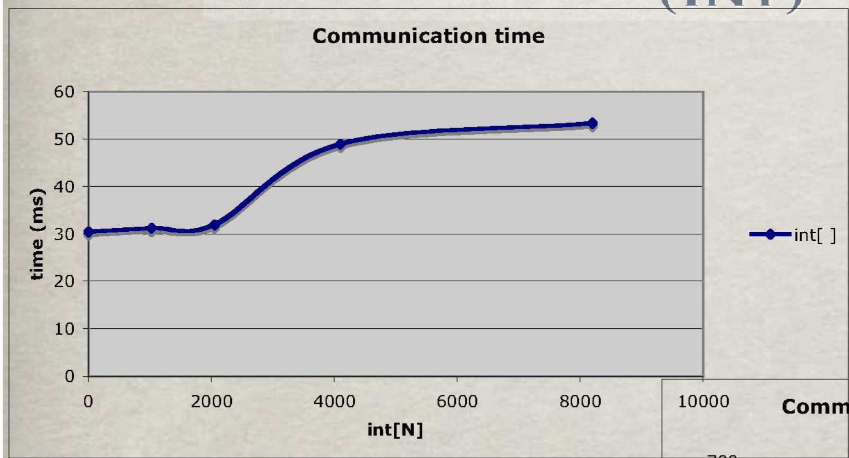
Proactive/Java appears quite heavyweight w.r.t. other approaches

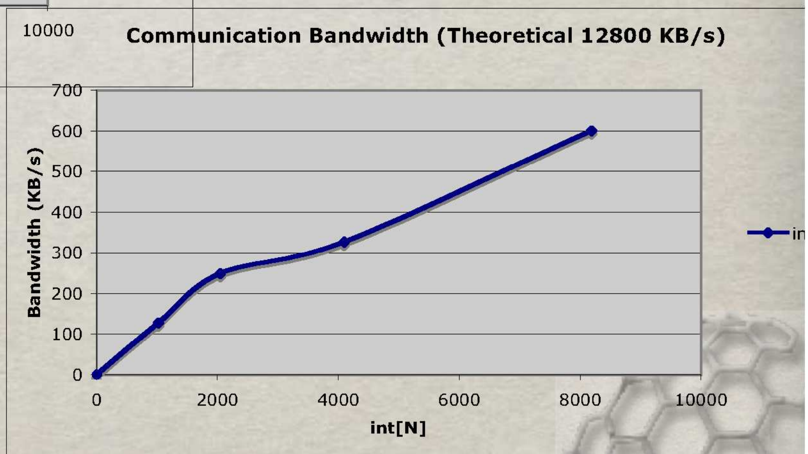
ASSIST/C++ overheads (ms)

M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo.
Dynamic reconfiguration of grid-aware applications in ASSIST.
Euro-Par 2005, vol. 3648 of LNCS, Lisboa, Portugal. Springer Verlag, August 2005.

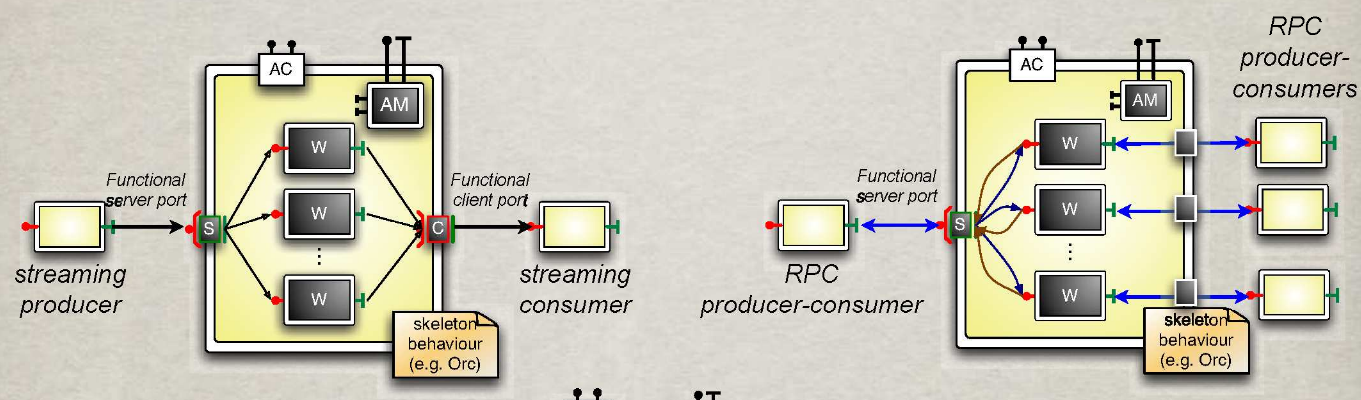
parmod kind	Data-parallel (with shared state)						Farm (without shared state)					
reconf. kind	add PEs			remove PEs			add PEs			remove PEs		
# of PEs involved	1→2	2→4	4→8	2→1	4→2	8→4	1→2	2→4	4→8	2→1	4→2	8→4
R_l on-barrier	1.2	1.6	2.3	0.8	1.4	3.7	–	–	–	–	–	–
R_l on-stream-item	4.7	12.0	33.9	3.9	6.5	19.1	~ 0	~ 0	~ 0	~ 0	~ 0	~ 0
R_t	24.4	30.5	36.6	21.2	35.3	43.5	24.0	32.7	48.6	17.1	21.6	31.9

Proactive Communication Time (Int)

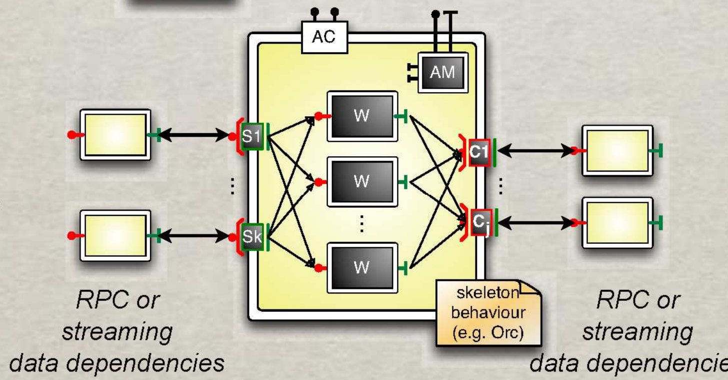




Variations and Flavours



or in general ...



and even more ...

Abstracting Out Variants

- *n* client and *y* server ports
  - synchronous and/or asynchronous
  - stream and/or RPC
  - programmable, possibly nondeterministic, relations among ports
    - wait for an item on port_A ***and/or*** one item on port_B
    - in general, any CSP expression
- But ... be careful, this is the **ASSIST** model
  - all features described above + distributed membrane + autonomicity, QoS contracts, limited hierarchy depth (i.e. 2)
  - sophisticated C++ implementation, language not easy to modify
- GCM should be *enough* expressive and *not too* complex
  - we consider ASSIST as the complexity asymptote

ASSIST References (2002-07 UNIPI+ISTI/CNR only)

1. M. Aldinucci, F. André, J. Buisson, S. Campa, M. Coppola, M. Danelutto, and C. Zoccolo. Parallel program/component adaptivity management. In G. R. Joubert, W. E. Nagel, F. J. Peters, O. Plata, P. Tirado, and E. Zapata, editors, Parallel Computing: Current & Future Issues of High-End Computing (Proc. of PARCO 2005, Malaga, Spain), volume 33 of NIC, pages 89–96, Germany, Dec. 2005. John von Neumann Institute for Computing.
2. M. Aldinucci, F. André, J. Buisson, S. Campa, M. Coppola, M. Danelutto, and C. Zoccolo. An abstract schema modeling adaptivity management. In S. Gorlatch and M. Danelutto, editors, Integrated Research in Grid Computing, CoreGRID. Springer, Dec. 2006.
3. M. Aldinucci, G. Antoniu, M. Danelutto, and M. Jan. Fault-tolerant data sharing for high-level grid programming: A hierarchical storage architecture. In M. Bubak, S. Gorlatch, and T. Priol, editors, Achievements in European Research on Grid Systems, CoreGRID Series, pages 67–81. Springer, Nov. 2007.
4. M. Aldinucci and A. Benoit. Automatic mapping of ASSIST applications using process algebra. In Proc. of HLPP2005: Intl. Workshop on High-Level Parallel Programming. Warwick University, Coventry, UK, July 2005.
5. M. Aldinucci and A. Benoit. Towards the automatic mapping of ASSIST applications for the grid. In S. Gorlatch and M. Danelutto, editors, Proc. of the Integrated Research in Grid Computing Workshop, volume TR-05-22, pages 59–68, Pisa, Italy, Nov. 2005. Università di Pisa, Dipartimento di Informatica.
6. M. Aldinucci and A. Benoit. Towards the automatic mapping of ASSIST applications for the grid. In S. Gorlatch and M. Danelutto, editors, Integrated Research in Grid Computing, CoreGRID, pages 73–87. Springer, Dec. 2006.
7. M. Aldinucci and A. Benoit. Automatic mapping of ASSIST applications using process algebra. Parallel Processing Letters, 2008.
8. M. Aldinucci, C. Bertolli, S. Campa, M. Coppola, M. Vanneschi, L. Veraldi, and C. Zoccolo. Self-configuring and self-optimising grid components in the gcm model and their ASSIST implementation. Technical Report TR-06-13, Università di Pisa, Dipartimento di Informatica, Italy, Aug. 2006.
9. M. Aldinucci, C. Bertolli, S. Campa, M. Coppola, M. Vanneschi, L. Veraldi, and C. Zoccolo. Self-configuring and self-optimizing grid components in the GCM model and their ASSIST implementation. In Proc of. HPC-GECO/Compframe (held in conjunction with HPDC-15), IEEE, pages 45–52, Paris, France, June 2006.
10. M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, M. Danelutto, P. Pesciullesi, R. Ravazzolo, M. Torquati, M. Vanneschi, and C. Zoccolo. ASSIST demo: a high level, high performance, portable, structured parallel programming environment at work. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, Proc. of 9th Intl. Euro-Par 2003 Parallel Processing, volume 2790 of LNCS, pages 1295–1300, Klagenfurt, Austria, Aug. 2003. Springer.
11. M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, M. Danelutto, P. Pesciullesi, R. Ravazzolo, M. Torquati, M. Vanneschi, and C. Zoccolo. A framework for experimenting with structure parallel programming environment design. In G. R. Joubert, W. E. Nagel, F. J. Peters, and W. V. Walter, editors, Parallel Computing: Software Technology, Algorithms, Architectures and Applications (Proc. of PARCO 2003, Dresden, Germany), volume 13 of Advances in Parallel Computing, pages 617–624, Germany, 2004. Elsevier.
12. M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, S. Magini, P. Pesciullesi, L. Potiti, R. Ravazzolo, M. Torquati, M. Vanneschi, and C. Zoccolo. The implementation of ASSIST, an environment for parallel and distributed programming. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, Proc. of 9th Intl Euro-Par 2003 Parallel Processing, volume 2790 of LNCS, pages 712–721, Klagenfurt, Austria, Aug. 2003. Springer.
13. M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, and C. Zoccolo. Components for high performance grid programming in grid.it. In V. Getov and T. Kielmann, editors, Proc. of the Intl. Workshop on Component Models and Systems for Grid Applications, CoreGRID series, pages 19–38, Saint-Malo, France, Jan. 2005. Springer.
14. M. Aldinucci, S. Campa, M. Coppola, S. Magini, P. Pesciullesi, L. Potiti, R. Ravazzolo, M. Torquati, and C. Zoccolo. Targeting heterogeneous architectures in ASSIST: Experimental results. In M. Danelutto, M. Vanneschi, and D. Laforenza, editors, Proc. of 10th Intl. Euro-Par 2004 Parallel Processing, volume 3149 of LNCS, pages 638–643. Springer, Aug. 2004.
15. M. Aldinucci, M. Coppola, S. Campa, M. Danelutto, M. Vanneschi, and C. Zoccolo. Structured implementation of component based grid programming environments. In V. Getov, D. Laforenza, and A. Reinefeld, editors, Future Generation Grids, CoreGRID series, pages 217–239. Springer, Nov. 2005.
16. M. Aldinucci, M. Coppola, M. Danelutto, N. Tonellotto, M. Vanneschi, and C. Zoccolo. High level grid programming with ASSIST. Computational Methods in Science and Technology, 12(1):21–32, 2006.
17. M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. ASSIST as a research framework for high-performance grid programming environments. Technical Report TR-04-09, Università di Pisa, Dipartimento di Informatica, Italy, Feb. 2004.
18. M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. ASSIST as a research framework for high-performance grid programming environments. In J. C. Cunha and O. F. Rana, editors, Grid Computing: Software environments and Tools, chapter 10, pages 230–256. Springer, Jan. 2006.
19. M. Aldinucci and M. Danelutto. Stream parallel skeleton optimization. In Proc. of PDCS: Intl. Conference on Parallel and Distributed Computing and Systems, pages 955–962, Cambridge, Massachusetts, USA, Nov. 1999. IASTED, ACTA press.
20. M. Aldinucci and M. Danelutto. An operational semantics for skeletons. In G. R. Joubert, W. E. Nagel, F. J. Peters, and W. V. Walter, editors, Parallel Computing: Software Technology, Algorithms, Architectures and Applications (Proc. of PARCO 2003, Dresden, Germany), volume 13 of Advances in Parallel Computing, pages 63–70, Germany, 2004. Elsevier.
21. M. Aldinucci and M. Danelutto. Algorithmic skeletons meeting grids. Parallel Computing, 32(7):449–462, 2006. DOI:10.1016/j.parco.2006.04.001.
22. M. Aldinucci and M. Danelutto. Skeleton based parallel programming: functional and parallel semantic in a single shot. Computer Languages, Systems and Structures, 2006. doi: 10.1016/j.cl.2006.07.004, in press.
23. M. Aldinucci, M. Danelutto, and J. Dünnweber. Optimization techniques for implementing parallel skeletons in grid environments. In S. Gorlatch, editor, Proc. of CMPP: Intl. Workshop on Constructive Methods for Parallel Programming, pages 35–47, Stirling, Scotland, UK, July 2004. Universität Münster, Germany.
24. M. Aldinucci, M. Danelutto, A. Paternesi, R. Ravazzolo, and M. Vanneschi. Building interoperable grid-aware ASSIST applications via WebServices. In G. R. Joubert, W. E. Nagel, F. J. Peters, O. Plata, P. Tirado, and E. Zapata, editors, Parallel Computing: Current & Future Issues of High-End Computing (Proc. of PARCO 2005, Malaga, Spain), volume 33 of NIC, pages 145–152, Germany, Dec. 2005. John von Neumann Institute for Computing.
25. M. Aldinucci, M. Danelutto, and M. Vanneschi. Autonomic QoS in ASSIST grid-aware components. In B. D. Martino and S. Venticinque, editors, Proc. of Intl. Euromicro PDP 2006: Parallel Distributed and network-based Processing, pages 221–230, Montbéliard, France, Feb. 2006. IEEE.
26. M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo. Dynamic reconfiguration of grid-aware applications in ASSIST. In J. C. Cunha and P. D. Medeiros, editors, Proc. of 11th Intl. Euro-Par 2005 Parallel Processing, volume 3648 of LNCS, pages 771–781. Springer, Aug. 2005.
27. R. Baraglia, M. Danelutto, D. Laforenza, S. Orlando, P. Palmerini, R. Perego, P. Pesciullesi, and M. Vanneschi. ASSISTConf: A grid configuration tool for the ASSIST parallel programming environment. In Proc. of Intl. Euromicro PDP: Parallel Distributed and network-based Processing, pages 193–200, Genova, Italy, Feb. 2003. IEEE.
28. P. D'Ambra, M. Danelutto, D. di Serafino, and M. Lapegna. Advanced environments for parallel and distributed applications: a view of current status. Parallel Computing, 28(12):1637–1662, 2002.
29. M. Danelutto. Dynamic run time support for skeletons. In E. H. D'Hollander, G. R. Joubert, F. J. Peters, and H. J. Sips, editors, Proc. of Intl. PARCO 99: Parallel Computing, Parallel Computing Fundamentals & Applications, pages 460–467. Imperial College Press, 1999.
30. M. Danelutto. Adaptive task farm implementation strategies. In Proc. of Intl. Euromicro PDP: Parallel Distributed and network-based Processing, pages 416–423, La Coruna, Spain, Feb. 2004. IEEE.
31. M. Danelutto. Irregularity handling via structured parallel programming. Intl. Journal of Computational Science and Engineering, 3-4, 2005.
32. M. Danelutto. QoS in parallel programming through application managers. In Proc. of Intl. Euromicro PDP: Parallel Distributed and network-based Processing, pages 282–289, Lugano, Switzerland, Feb. 2005. IEEE.
33. M. Danelutto, C. Migliore, and C. Pantaleo. An alternative implementation schema for ASSIST parmod. In Proc. of Intl. Euromicro PDP: Parallel Distributed and network-based Processing, pages 56–63, Montbéliard, France, Feb. 2006. IEEE.
34. M. Danelutto and M. Vanneschi. A RISC approach to Grid. In B. D. Martino, J. Dongarra, A. Hoisie, L. T. Yang, and H. Zima, editors, Engineering the grid, chapter 8. ASP press, Jan. 2006.
35. M. Danelutto, M. Vanneschi, C. Zoccolo, N. Tonellotto, S. Orlando, R. Baraglia, T. Fagni, D. Laforenza, and A. Paccosi. HPC application execution on grids. In V. Getov, D. Laforenza, and A. Reinefeld, editors, Future Generation Grids, CoreGRID series, pages 263–282. Springer, Nov. 2005.
36. D. Laforenza and M. Vanneschi. Grid.it - next generation grid platforms and their applications. ERCIM News, 59:60–61, Oct. 2004.
37. S. Magini, P. Pesciullesi, and C. Zoccolo. Parallel software interoperability by means of CORBA in the ASSIST programming environment. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, Proc. of 9th Intl Euro-Par 2003 Parallel Processing, volume 2790 of LNCS, pages 679–688, Klagenfurt, Austria, Aug. 2003. Springer.
38. I. Merelli, L. Milanesi, D. D'Agostino, A. Clematis, M. Vanneschi, and M. Danelutto. Using parallel isosurface extraction in superficial molecular modeling. In 1st Intl. Conference on Distributed Frameworks for Multimedia Applications (DFMA 2005), pages 288–294, Besançon, France, 2005. IEEE.
39. N. Tonellotto, D. Laforenza, M. Danelutto, M. Vanneschi, and C. Zoccolo. A performance model for stream-based computations. In P. D'Ambra and M. R. Guarracino, editors, Proc. of Intl. Euromicro PDP 2007: Parallel Distributed and network-based Processing, pages 91–96, Napoli, Italia, Feb. 2007. IEEE.
40. M. Vanneschi. Heterogeneous HPC environments. In D. J. Pritchard and J. Reeve, editors, Proc. of 4th Intl. Euro-Par '98 Parallel Processing, volume 1470 of LNCS, pages 21–34, Southampton, UK, 1998. Springer.
41. M. Vanneschi. The programming model of ASSIST, an environment for parallel and distributed portable applications. Parallel Computing, 28(12):1709–1732, Dec. 2002.
42. M. Vanneschi. ASSIST high-performance programming environment: Application experiences and grid evolution. In J. Dongarra, D. Laforenza, and S. Orlando, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface,10th European PVM/MPI Users' Group Meeting, Venice, Italy, September 29 - October 2, 2003, Proceedings, volume 2840 of LNCS, pages 24–26, Venice, Italy, 2003. Springer.

BeSke, GCM, and ORC References GridCOMP or CoreGrid related (2007-08 UNIPI+ISTI/CNR only)

1. M. Aldinucci, S. Campa, M. Danelutto, P. Dazzi, P. Kilpatrick, and N. Tonellotto. Management in distributed systems: a semi-formal approach. In Proc. of Intl. Euromicro PDP 2008: Parallel Distributed and network-based Processing, Toulouse, France, Feb. 2008. IEEE. To appear.
2. M. Aldinucci, S. Campa, M. Danelutto, P. Dazzi, P. Kilpatrick, D. Laforenza, and N. Tonellotto. Behavioural skeletons for component autonomic management on grids. In CoreGRID Workshop on Grid Programming Model, Grid and P2P Systems Architecture, Grid Systems, Tools and Environments, Heraklion, Creete, Greece, June 2007.
3. M. Aldinucci, M. Danelutto, and P. Kilpatrick. Adding metadata to orc to support reasoning about grid programming. In T. Priol and M. Vanneschi, editors, Towards Next Generation Grids (Proc. of the CoreGRID Symposium 2007), CoreGRID series, pages 205–214, Rennes, France, Sept. 2007. Springer.
4. M. Aldinucci, M. Danelutto, and P. Kilpatrick. A framework for prototyping and reasoning about grid systems. In G. R. Joubert, C. Bischof, F. Peters, T. Lippert, M. Bücker, P. Gibbon, and B. Mohr, editors, Parallel Computing: Architectures, Algorithms and Applications (Proc. of PARCO 2007, Jülich, Germany), NIC, Germany, Sept. 2007. John von Neumann Institute for Computing.
5. M. Aldinucci, M. Danelutto, and P. Kilpatrick. Management in distributed systems: a semi-fomal approach. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, Proc. of 13th Intl. Euro-Par 2007 Parallel Processing, volume 4641 of LNCS, pages 651—661, Rennes, France, Aug. 2007. Springer.
6. M. Aldinucci, M. Danelutto, and P. Kilpatrick. Orc + metadata supporting grid programming. Technical Report TR-07-10, Università di Pisa, Dipartimento di Informatica, May 2007.
7. M. Danelutto, M. Aldinucci, and P. Kilpatrick. Prototyping and reasoning about distributed systems: an orc based framework. Technical Report TR-0102, Institute on Programming Model, CoreGRID - Network of Excellence, Aug. 2007.
8. P. Kilpatrick, M. Danelutto, and M. Aldinucci. Deriving grid applications from abstract models. Technical Report TR-0085, Institute on Programming Model, CoreGRID - Network of Excellence, Apr. 2007.
9. M. Aldinucci, C. Bertolli, S. Campa, M. Coppola, M. Vanneschi, L. Veraldi, and C. Zoccolo. Self-configuring and self-optimizing grid components in the GCM model and their ASSIST implementation. In Proc of. HPC-GECO/Compframe (held in conjunction with HPDC-15), IEEE, pages 45–52, Paris, France, June 2006.

Conclusions

- Behavioural Skeletons
  - templates with built-in management for the App designer
  - methodology for the skeleton designer
    - management can be changed/refined
    - just prove your own management is correct against skeleton functional description
  - can be freely mixed with standard GCM components
    - because once instanced, they are standard
- Already implemented on GCM
  - not happy about GCM runtime performances (can be improved)
    - We also implemented in ASSIST with different performances

Session 2 - User presentations

GridCOMP Workshop
October 31st, 2007
CNIC, Beijing, China

Wrapping legacy PL/SQL enterprise code using GCM

ATOS ORIGIN

31-oct-2007 – Beijing, China

Fabio Tumiatti

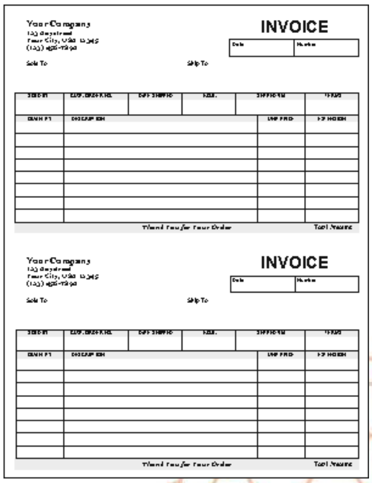
fabio.tumiatti@atosorigin.com

Index

- Use case summary
- Preliminary architecture
- Components Description

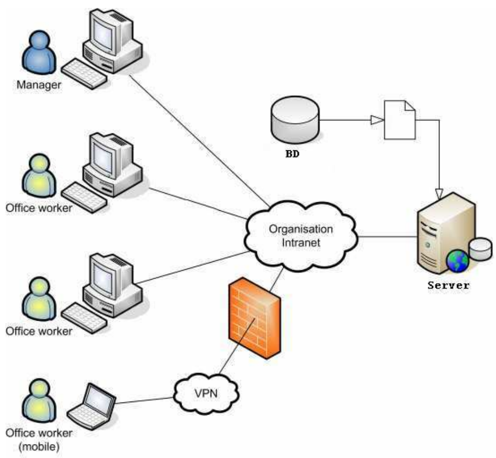
Use case summary (1/5)

- Computing of DSO value
  - The DSO (**D**ays **S**ales **O**utstanding) is the mean time that clients delay to pay an invoice to Atos
  - Information is needed by several internal departments as much updated as possible
  - Process lasts about 4 hours to complete (over 6.000 clients)



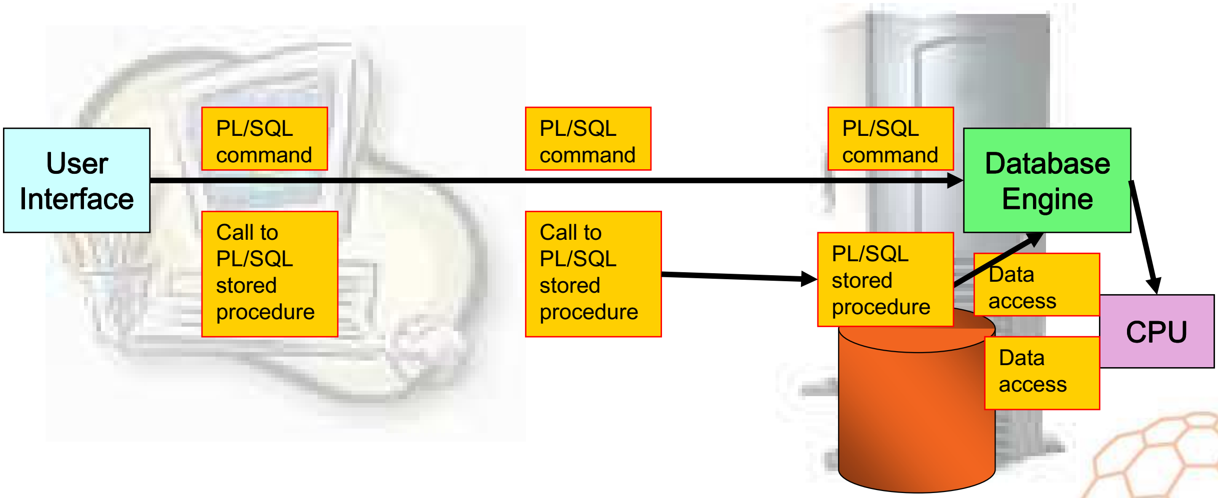
Use case summary (2/5)

- The DSO application is based on a client/server application
  - A Graphical User Interface
    - to input data or parameters needed for the computation
  - Some PL/SQL processes
    - to access the data stored in the database and process them to compute the results
  - The database
    - to store the data



Use case summary (3/5)

- Heavy processes written in PL/SQL (Oracle Stored Procedures)



Use case summary (4/5)

- The PL/SQL code
  - Must avoid (or minimize at maximum) the rewriting of PL/SQL procedures to avoid re-testing the critical business code
  - Split the PL/SQL code into independent sub-programs
    - Read / Write / Compute
  - Organize the sub-programs between the master and node databases

Use case summary (5/5)

GridCOMP

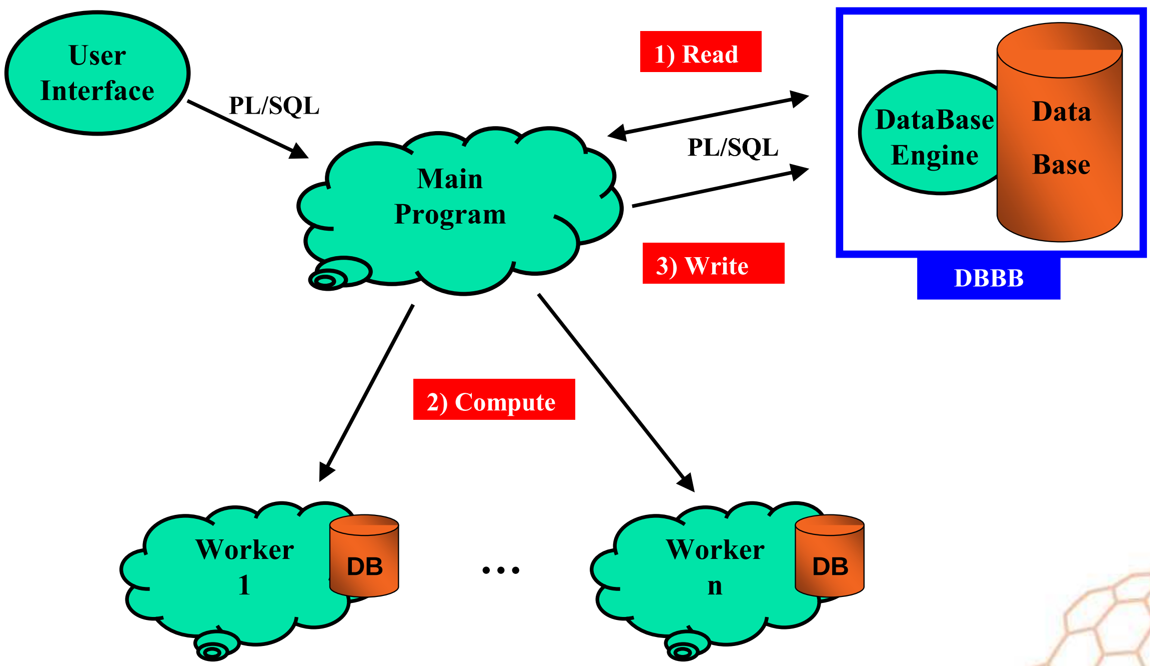
Before

- Process lasts about 4 hours to complete (over 6.000 clients)
- Information not updated frequently

After

- Reduce execution time
- Information updated more frequently
- Maintain/reduce infrastructure costs

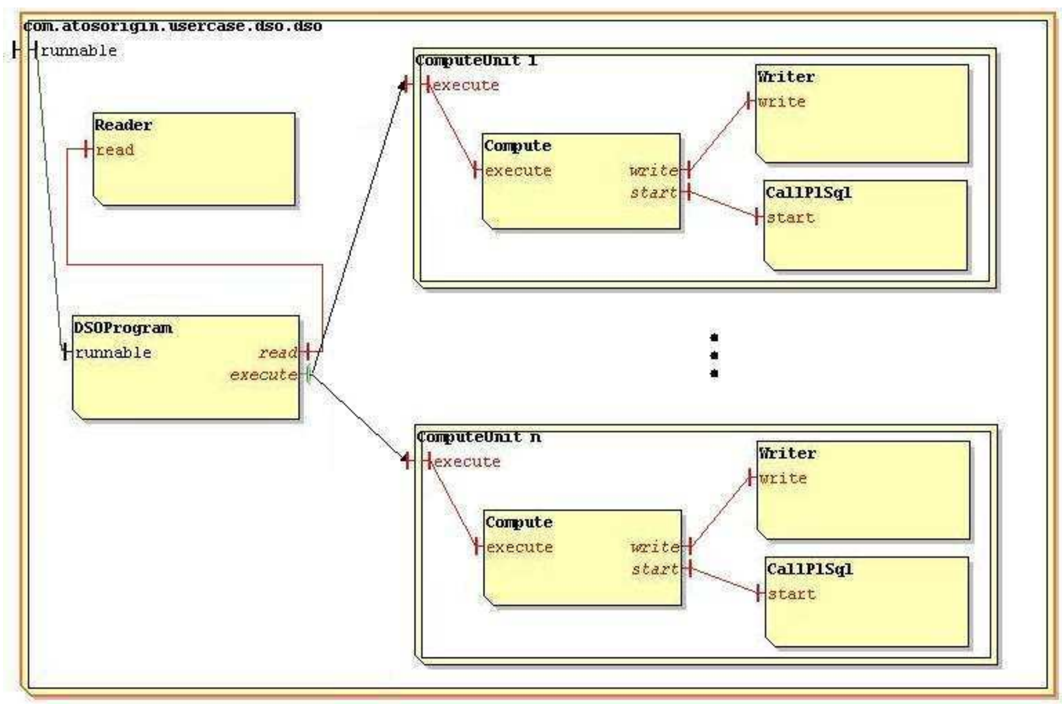
Preliminary architecture (1/2)



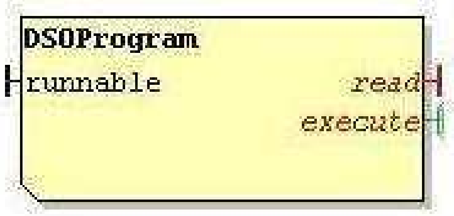
Preliminary architecture (2/2)

- Master database
  - Oracle Standard Edition or Enterprise Edition
  - store all data and the packages (functions)
- Node database
  - Oracle Database 10g Express Edition
    - Free of charge
    - Limitations:
      - **Memory**: only 1GB of RAM
      - **CPU**: only use one CPU
      - **Database**: only a single XE can run on any given computer
      - **Disk space**: a 4GB limit
  - store part of the PL/SQL code and some data

Preliminary architecture – GCM Components



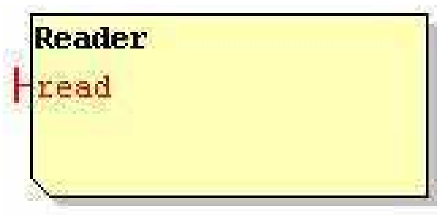
Components Description (1/6)



- The DSOProgram is the master component of the prototype
  - program workflow
  - 2 client's interfaces called *read* and *execute*
    - *execute* – multicast client interface using *Broadcast* and *Round_robin* dispatch mode

```
public interface OurTaskMulticast extends Serializable {
 public List<BooleanWrapper>
    compute(@ParamDispatchMetadata(mode=ParamDispatchMode.ROUND_ROBIN) List<List<String>>
    clients, @ParamDispatchMetadata(mode=ParamDispatchMode.BROADCAST) List<String> dates);
}
```

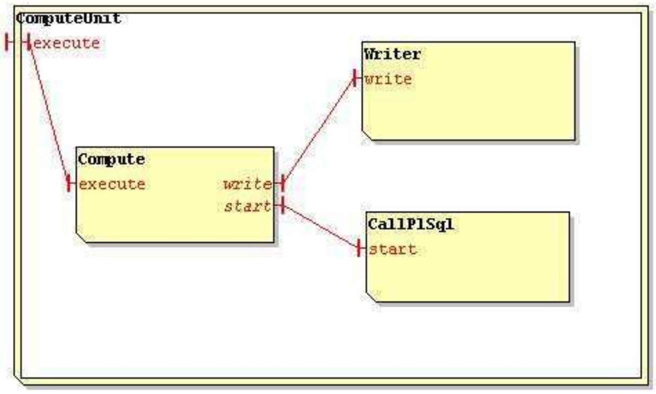

Components Description (2/6)



- The Reader component offers the functionality to connect to the master database
  - gets the list of client's ids to be sent as parameters to the PL/SQL code

```
public interface Reader {
    String[] getClients(String clientId, String groupId);
}
```

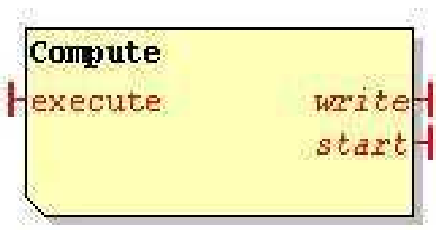
Components Description (3/6)



- The ComputeUnit component is responsible for the tasks execution on the slaves
  - offers a *execute* server interface

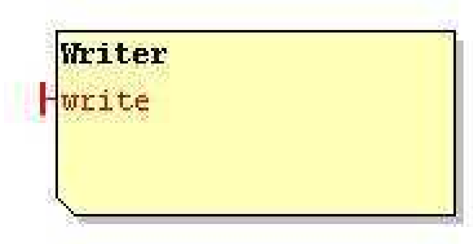
```
public interface OurTask extends Serializable {
 public BooleanWrapper compute(List<String>  clients, List<String> dates);
}
```


Components Description (4/6)



- The Compute component offers the functionality to receive the tasks from the ComputeUnit and execute them
  - receives the list of client's ids and sends it to the Writer component to insert it on the salve database.
  - calls the CallPlSql component to execute the PL/SQL code

Components Description (5/6)



- The Writer component offers the functionality to write on the node database the list of client's ids to be processed by the PL/SQL code

```
public interface Writer {
    public BooleanWrapper insertClients(String[] clients, String start_date, String
    end_date);
}
```


Components Description (6/6)

- The CallPlSql component offers the functionality of wrapping PL/SQL code.
  - calls an Oracle stored procedure on the slave database to execute the PL/SQL code

```
public interface CallPlSql {
    public BooleanWrapper executePlSql();
}
```

Conclusions

- Analyze the code:
  - Right distribution of the PL/SQL code between the master DB and the nodes to use with GCM
- Need to avoid:
  - Rewriting the PL/SQL code

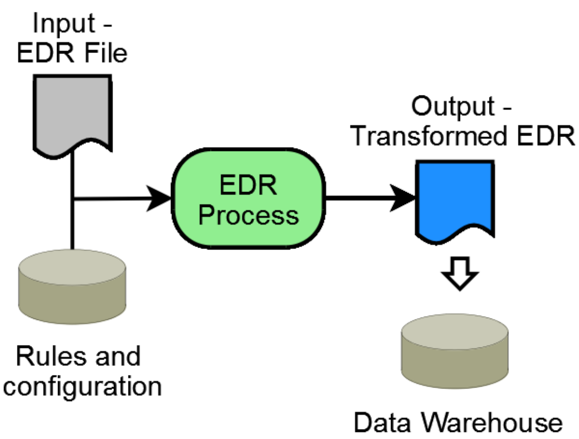
- Benefits:
  - Reduction of execution time
  - Reduction of Infrastructure costs
  - Platform independency (Linux, Windows, …)

Telecom Computing Application (EDR Processor)

Gastón Freire

gfreire@gridsystems.com

The Problem (I)



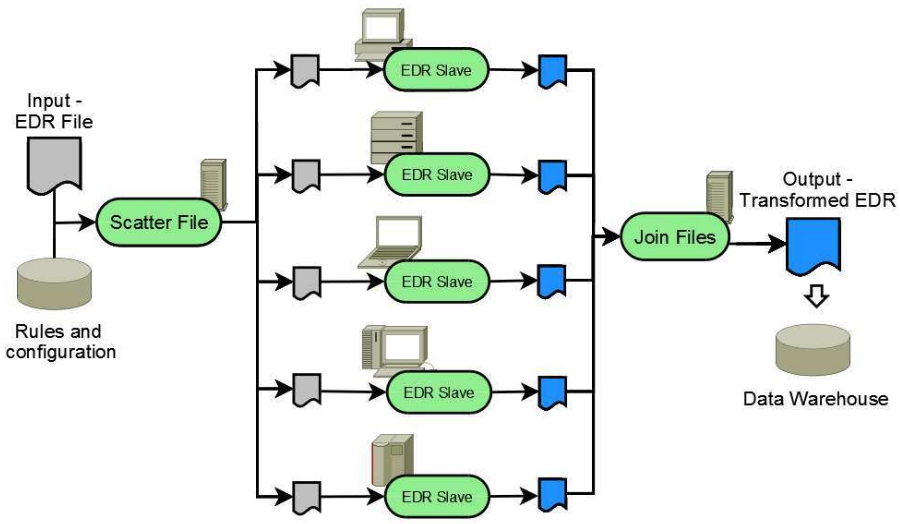
- Analyze all the data about network services
- EDRs (Extended Data Records) files contains data related to calls and other services (SMS, WAP).
- These files are:
  - processed continuously
  - stored in a Data Warehouse
  - accessed by several business processes

The Problem (II)

- EDR processing ⇨ ETL (Extract, Transform and Load).
- ETL means
  - collect data,
  - process it,
  - feed it to a data warehouse or database
- Expansion of telecom services
  - More computational needs
  - Traditional ETL processing is not enough
  - More EDRs must be processed in less time.

The solution: GridCOMP (I)

- Using GridCOMP we can distribute the transformation effort



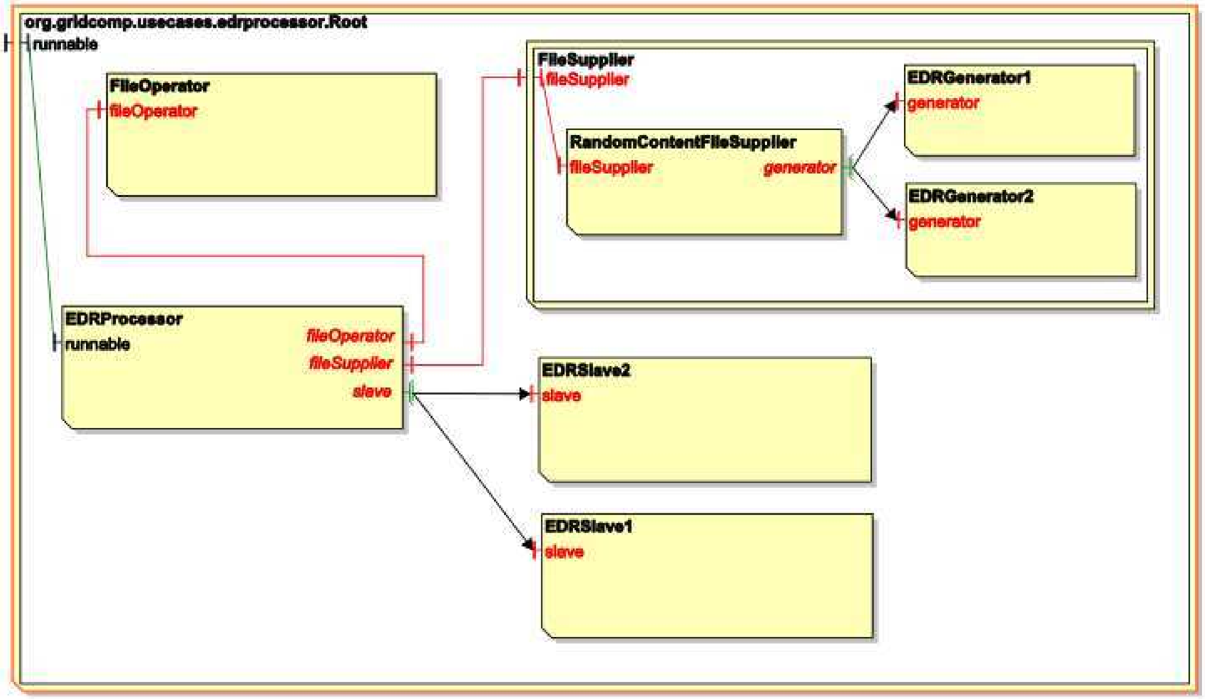
The Solution: GridCOMP (II)

- What GridCOMP offers to this application:
  - A grid-computing component-based model (GCM)
    - 100% Java™
    - 100% portable
  - A Grid IDE to design the architecture of our application.
  - Composition of components
    - Follow a top-down design of the application
    - Easy to reuse code
  - Collective interfaces
    - Abstract and hide the complexity of distributed computing
  - Autonomic component management
    - Provides fault tolerance and load balancing

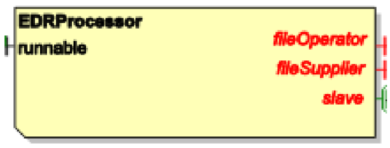
The Solution: GridCOMP (III)

- Benefits:
  - Reduced processing time
    - more complex processes
    - in less time
  - Redundancy and fault-tolerance
    - improves the quality of the service
  - Cost
    - Use existing hardware
    - Use low-profile machines
  - Scalability
    - Easy to add more power
    - No need to change the application to scale out

EDR Processing – Current Architecture

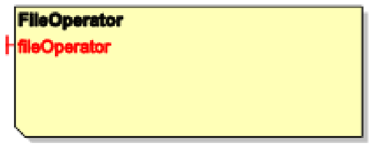


EDR Processing – Components Description



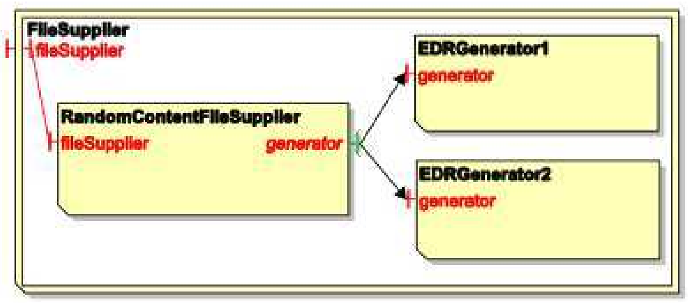
- The **EDRProcessor** acts as the master component
  - Obtains the EDR file to be processed from the `fileSupplier`
  - Scatters the file using the `fileOperator`
  - Processes the chunks using the `slave` multicast interface
  - Joins the partial results using the `fileOperator`

EDR Processing – Components Description



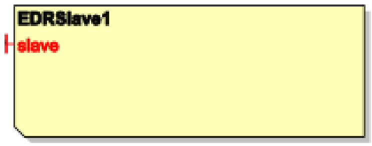
- The **FileOperator** component offers the functionality to scatter and join files

EDR Processing – Components Description



- The **FileSupplier** supplies the EDR files to be processed.
- The current implementation randomly generates the content of the EDR files

EDR Processing – Components Description



- Each one of the **EDRSlave** components:
  - Downloads the corresponding chunk of the EDR file
  - Processes each record, generating a partial result file
  - Uploads the partial result file

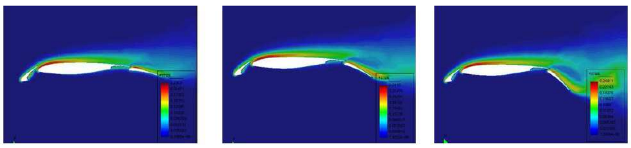
Demonstration

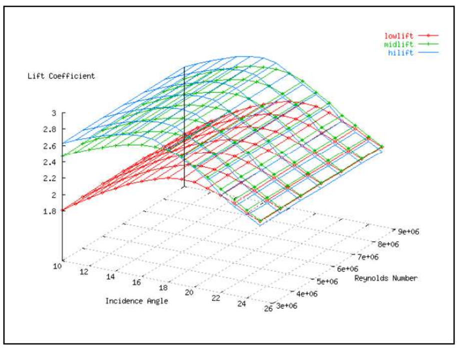
Scientific Computing Application (Wing Design)

Gastón Freire

gfreire@gridsystems.com

The Problem (I)





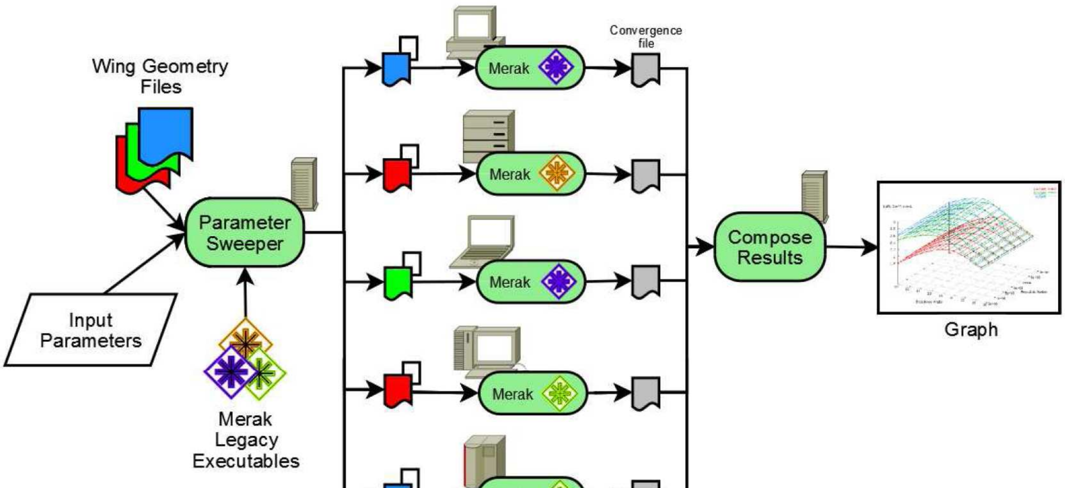
- Aerospace sector
- Compare the aerodynamic performance of different wing configurations
- Legacy fortran-77 program: Merak
  - Windows, Linux, Solaris
- Merak receives
  - a wing configuration,
  - an incidence angle
  - a Reynolds number
- Merak produces a convergence file that contains
  - lift,
  - drag,
  - residuals, etc.

The Problem (II)

- Perform a parametric sweep on
  - A set of wing geometries
  - A range of incidence angles
  - A range of values for the Reynolds Number
- Display the results as a graph
- Each simulation consumes CPU heavily
  - Parallelize computation to achieve useful response times (minutes)

The Solution: GridCOMP (I)

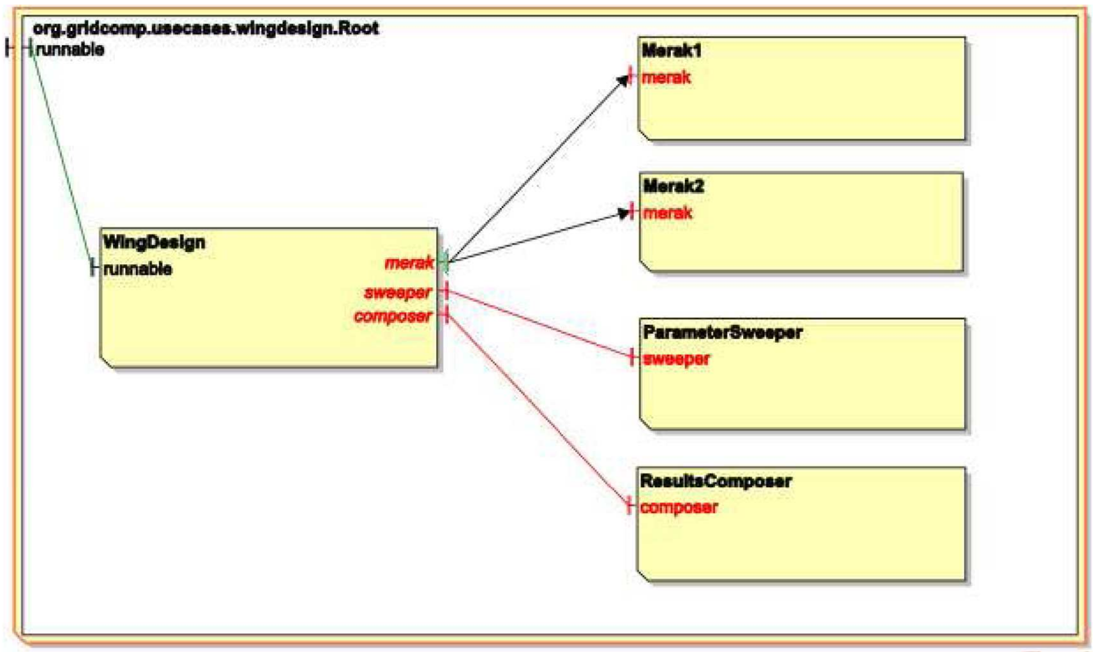
- Using GridCOMP we can distribute the computation effort



The Solution: GridCOMP (II)

- What GridCOMP offers to this application:
  - **Legacy code wrapping support.**
    - Legacy application ⇨ Component.
  - Composition of components
    - Combine new and pre-existing components.
  - Collective interfaces
    - Abstract and hide the complexity of distributed computing
  - Autonomic component management
    - Provides fault tolerance and load balancing
  - Tools to design the architecture of our grid application (GIDE).

Wing Design – Current Architecture



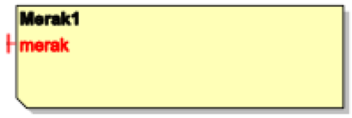
Wing Design – Components description

- The **WingDesign** acts as the master component
  - Obtains the list of parameter combinations to be evaluated using the `sweeper` interface
  - Processes the list of parameter combinations using the `merak` multicast interface
  - Composes the graph displaying the results using the `composer` interface

Wing Design – Components description

- The **ParameterSweeper** generates parameter combinations performing the Cartesian product of:
  - The range of incidence angles
  - The range of Reynolds numbers
  - The range of wing configurations

Wing Design – Components description



- The **Merak** component wraps the legacy application:
  - Downloads the proper executable files
  - Processes each request
    - prepare input parameters,
    - invoke executable,
    - transfer result file
  - Deletes temporary files on finishing

Wing Design – Components description

- The **ResultsComposer:**
  - Parses the result files
  - Extracts the interesting values
  - Composes a `.plo` file with the extracted data
  - Invokes `gnuplot` to plot the graph

Demonstration

Grid programming with components: an advanced COMPonent platform for an effective invisible grid

Legacy code wrapping, Interoperation with CGSP and Bioinformatics application

Weiyuan Huang
Tsinghua University
huang-wy05@mails.tsinghua.edu.cn

Agenda

- Legacy code and interoperability
- Interoperability with CGSP
- CGSP as Deployment protocol
- Example: Bioinformatics computing

Agenda

- Legacy code and interoperability
- Interoperability with CGSP
- CGSP as Deployment protocol
- Example: Bioinformatics computing

Legacy Code Wrapping and Interoperability

- The purpose of this task is to develop techniques and methods for turning those legacy codes into components. It includes:
  - Extend ADL
    - Add some specific tags into the standard ADL to take into account the legacy code specificities
  - Java API
    - Define the standard API for wrapping the legacy codes to components
    - Define the needed server and client interfaces to manipulate and control the legacy code

What we have been done

- At present, we have done most of the Java packages and implemented the design primarily.
  - Extend the ADL and define the DTD
  - Implement the Java packages
    - Legacy code process
    - Execution status manager
    - ADL parser
    - Define interface to manipulate and control the code
    - Judge the execution environment of the legacy code
    - ……

Java package

- net.coregrid.gcm.legacyComponent
  - *wrap the legacy code to Component*
- net.coregrid.gcm.legacyComponent.process
  - *Implement the running process of the legacy code*
- net.coregrid.gcm.legacyComponent.legacyCode
  - *Implement the interface to manipulate and control the code*
- net.coregrid.gcm.legacyComponent.ADL
  - *Define the class to store the legacy code's information from the ADL*
- net.coregrid.gcm.legacyComponent.ADL.paser
  - *Define the parser of the legacy code ADL*
- net.coregrid.gcm.legacyComponent.relatedfile
  - *Set the access permission of the related files*
- net.coregrid.gcm.legacyComponent.resource
  - *Judge the execution environment of the legacy code*

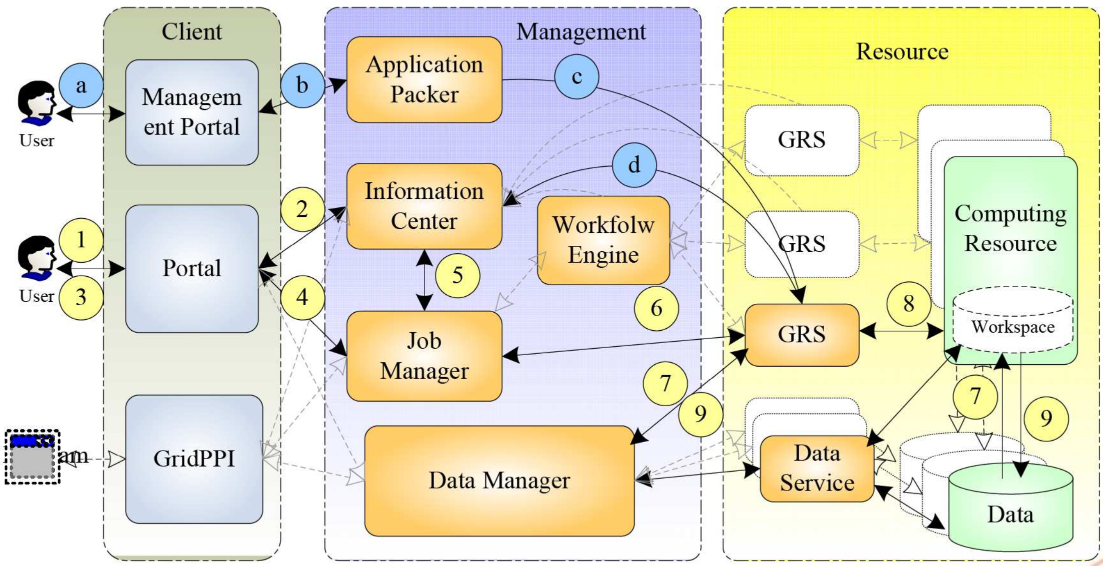
Agenda

- Legacy code and interoperability
- Interoperability with CGSP
- Treat CGSP as Deployment protocol
- Example: Bioinformatics computing

Brief Introduction

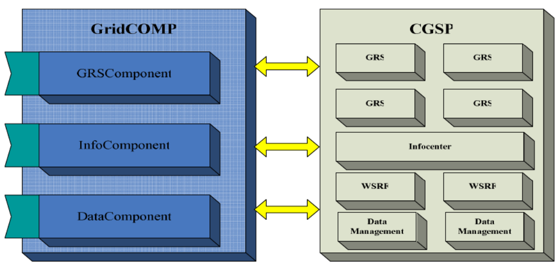
- CGSP is a grid middleware developed for the construction and evolution of ChinaGrid;
- Based on OGSA, CGSP is developed according to the latest grid specification WSRF
- CGSP support highly localized requirement and autonomy requirement
- Scalability of CGSP satisfies the demand of expansion of grid system
- CGSP guarantees the integrity and uniformity of grid platform by a global monitoring system

The workflow of CGSP



GCM interoperate with CGSP

- GridCOMP interact with CGSP
  create three basic component of CGSP
    GRS Component
    Info Component
    Data Component



GCM interoperate with CGSP

The process of submit a CGSP Job

1. Upload the input data using Data Component
2. Pack a GRS Job using GRS Component
3. Upload the executable program
4. Fill in all the parameters of GRS Job
5. Query the GRS Job using Info Component
6. Invoke the Job
7. Get the status of the Job
8. Get the result

The following pseudo code is an example of submit a CGSP Job. This job is a bio-information example CAP3 to compare DNA sequences.

GCM interoperate with CGSP

GRS Component

GRS Component packs a executable program and interacts with all the modules of CGSP to finish a job.

Functions GRS Component provides:

1. Packing: pack a executable program as a WSRF
2. Upload Program: upload the executable program
3. Fill Parameter: use the data from data space as the input data
4. Invoke: invoke the GRS Job
5. Get Status: get the status of the GRS Job submitted
6. Get Results: get the result of the job or “error”

GCM interoperate with CGSP

Info Component

Info Component gets all the information of deployed application of CGSP and returns them as GRS Components. It also provides interface to register a WSRF to the CGSP.

Functions Info Components provides:

1. Query: get the right applications developers require and return them as GRS Components
2. Register: register a WSRF to the CGSP

GCM interoperate with CGSP

Data Component

Data Component provider developers with upload/download interface to interact with data space of CGSP.

Data Component provides:

1. Upload: upload the data to the data space of CGSP
2. Download: download data from data space of CGSP
3. List: get all data of data space

Sample Code

```
GRSCOMP grsComp = new GRSCOMP(url);
DataCOMP dataComp = new DataCOMP(url);
InfoCOMP infoComp = new InfoCOMP(url);
VirtualFileHandler vfh = dataComp.upload("/home/user/sequence.txt", "vs://admin/input/");
grsComp.pack("cap3GRSApp", "cap3", "inputfile", "-a 20", "output");
grsComp.uploadProgram("/home/user/cap3");
grsComp.fillinParam(vfh, "-a 20", "vs://admin/output");
grsComp.registerToInfo();
GRSCOMP gc = infoComp.query("cap3GRSApp");
gc.invoke();
......
JobStatus jobStatus = gc.getStatus();

while(jobStatus != RUNNING) {
       jobStatus = gc.getStatus();
}

if (jobStatus == FINISHED) {
       File[] files = gc.getResult();
}
else {
       System.out.println("error");
}
```


Agenda

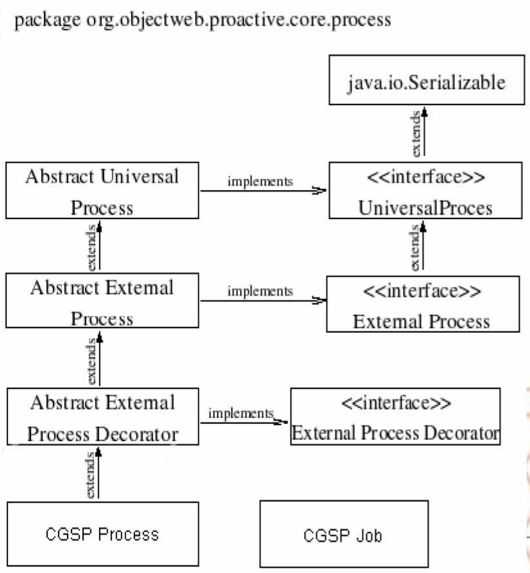
- Legacy code and interoperability
- Interoperability with CGSP
- CGSP as Deployment protocol
- Example: Bioinformatics computing

CGSP as Deployment Protocol

- Treat CGSP as a deployment protocol. Thus, there are 2 things we have to do:
  - Create the CGSP Java Process
  - Change XML Descriptor

CGSP Java Process

- CGSPProcess extends AbstractExternalProcessDecorator
- CGSPJob describes the CGSP Job



XML Descriptor

- Modifications to XML Deployment Descriptor Schema
- XML Parsing handler

Modifications to XSD

- First, add the “cgspProcess” to the “ProcessDefinitionType”.

```
<xs:complexType name="ProcessDefinitionType">
   <xs:choice>
      …
      <xs:element name="cgspProcess" type="cgspProcessType"/>
   </xs:choice>
   <xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
```

- Define “cgspProcessType”

```
<xs:complexType name="cgspProcessType" mixed="true">
  <xs:all>
    <xs:element ref="environment" minOccurs="0"/>
    <xs:element ref="processReference"/>
    <xs:element name="cgspOption" type="cgspOptionType" minOccurs="0"/>
  </xs:all>
  <xs:attribute name="class" type="xs:string" use="required"
fixed="org.objectweb.proactive.core.process.cgsp.CGSPProcess"/>
  <xs:attribute name="hostname" type="xs:string" use="required"/>
  <xs:attribute name="closeStream" type="CloseStreamType" use="optional"/>
  <xs:attribute name="queue" type="xs:string" use="optional"/>
</xs:complexType>
```

```
<xs:complexType name="cgspOptionType" mixed="true">
  <xs:all>
    <xs:element name="Count" type="PosintOrVariableType" minOccurs="0"/>
    <xs:element name="OutputFile" type="TextOrVariableType" minOccurs="0"/>
    <xs:element name="ErrorFile" type="TextOrVariableType" minOccurs="0"/>
  </xs:all>
</xs:complexType>
```

XML Parsing Handler

- Class “ProActiveDescriptorContants”. Add CGSP tag names used within XML files

Contants	Value
CGSP_PROCESS_TAG	“cgspProcess”
CGSP_OPTIONS_TAG	“cgspOption”

- Class “ProcessDefinitionHandler”. Add inner class “CGSPProcessHandler”, then register it.

```
// Add inner class “CGSPProcessHandler”
public class ProActiveDescriptorContants {
      …
      protected class CGSPProcessHandler extends ProcessHandler {
      …
            protected class CGSPOptionHandler extends PassiveCompositeUnmarshaller {
            }
      …
      }
}
```

```
// Register
this.addHandler(CGSP_PROCESS_TAG, new
CGSPProcessHandler(proActiveDescriptor));
```


Example

- Deploy XML

```
<cgspProcess hostname="cgsp_node">
    <processReference refid="localJVM1" />
    <cgspOption>
        <Count>10</Count>
        <errorFile base="root" relpath="${USER_HOME}/error.txt"/>
    </cgspOption>
</cgspProcess>
```

- Test Program

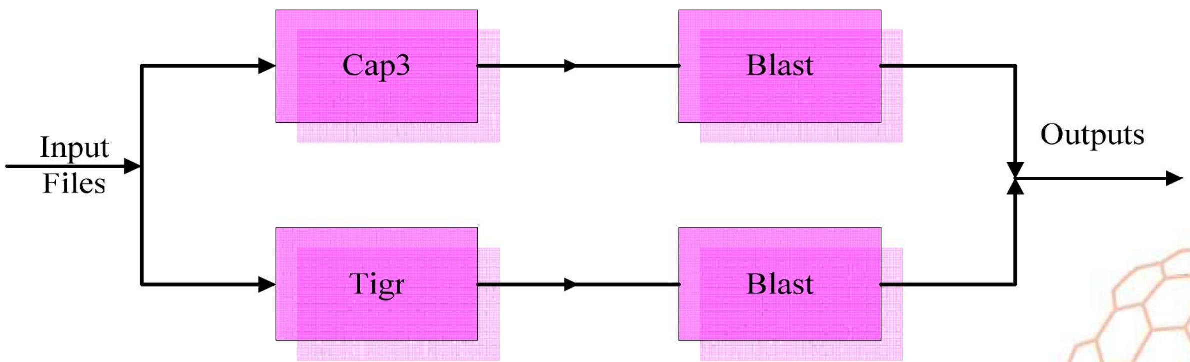
```
public class Test {
    public static void main(String[] args) {

        String command = "(executable = ${JAVA_HOME}/bin/java )(count=5)";

        CGSPJob Job1 = new CGSPJob();

        Job1.Run(command);

    }
}
```

Agenda

- Legacy code and interoperability
- Interoperability with CGSP
- CGSP as Deployment Protocol
- Example: Bioinformatics computing

WP5: Bioinformatics Computing (Workflow)

- Belong to Scientific Computing Application
- This Bioinformatics program can be divided into 3 sub program{CAP3 , Tigr (Gene Sequence Assembly Tool) ,Blast (Sequence Alignment Tool)}, and they have natural parallel characteristics.



Use case with Component

- Work flow:

(1)Mapping & get Nodes

(2)File transfer

(3)Legacy code ->primitive Component

(4) primitive Component ->Composite Component(CC)

(5)Component deployment

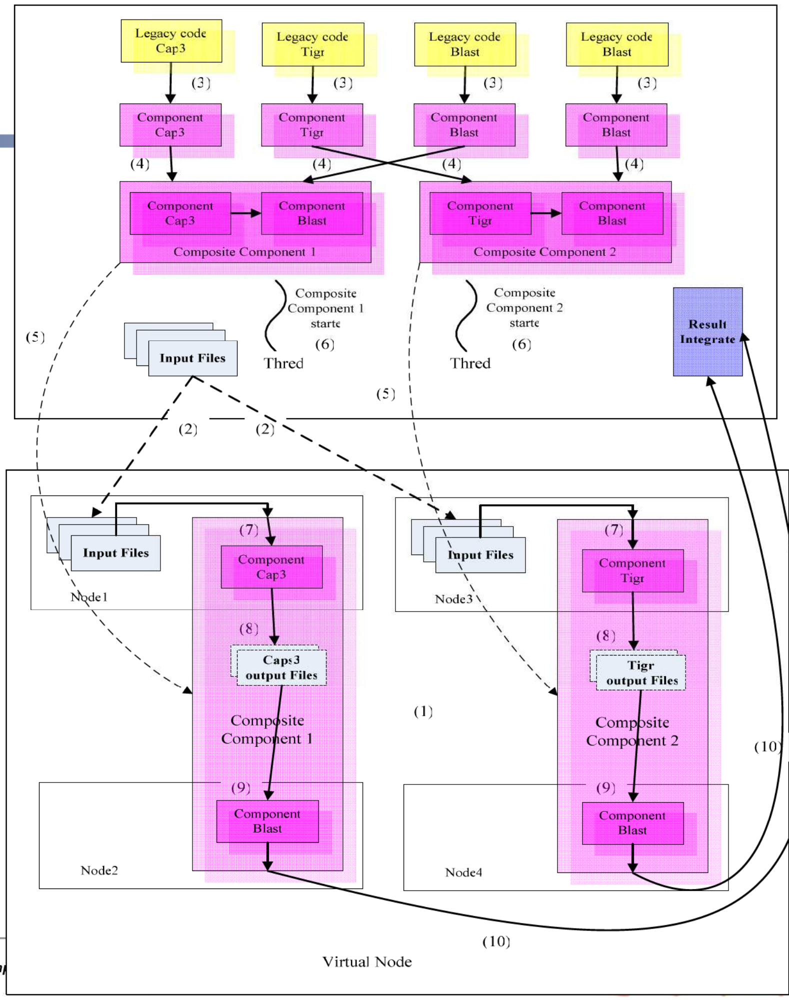
(6) Composite Component &2 are started parallel

(7)In Node1&3, the component Cap3 in CC 1 and component Tigr in CC2 are working parallel.

(8)When Cap3&Tigr finish their work ,they transfer the output files to node2&4

(9)The two Component Blast on node 2&4 begin to work

(10)Return each result of the CC to the Result Integrate



Thanks

Load-Balancing for Multicast Interfaces

Matthieu Morel

University of Chile

Problem statement

1. Computational speedup through parallel resources
2. Paradigms
   - Tightly coupled (SPMD)
   - Divide & conquer
   - Service composition, Workflows
   - **Embarrassingly parallel** (some GridCOMP use cases)

- Efficiency depends on:
  - Modeling of the problem
  - Partitioning - size
  - Infrastructure
- Infrastructure is often :
  - Volatile
  - Heterogeneous

Philosophy of ProActive / GCM

- offer component-based programming
  - Separation functional - non functional
  - Inversion of control
  - Customization (controllers)
- provide common facilities
  - Deployment
  - Assembly
  - Communication

Solutions for embarrassingly parallel problems

- Dedicated schedulers
  - Ex: *ourgrid* scheduler

    ☹ Focus on task allocation
    Coarse grained tasks
- Alternative:

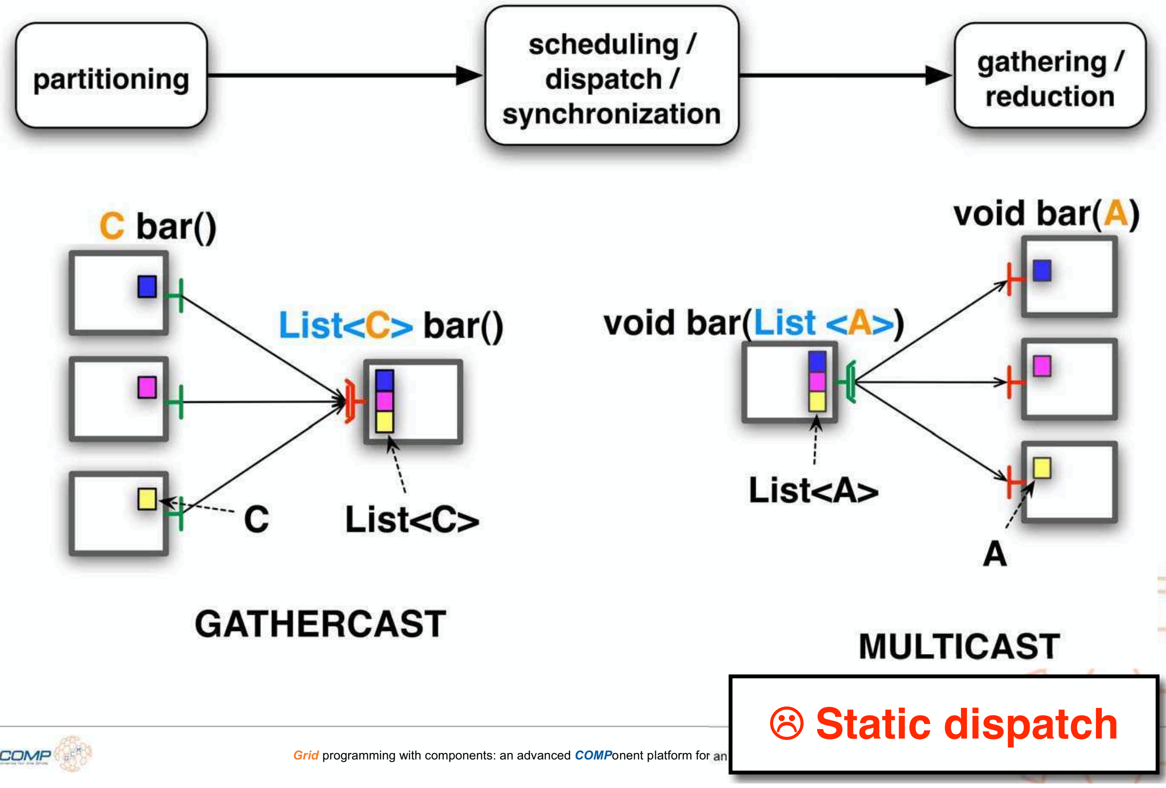
  ☺ Focus on the problem

  ☺ Structured assembly of components

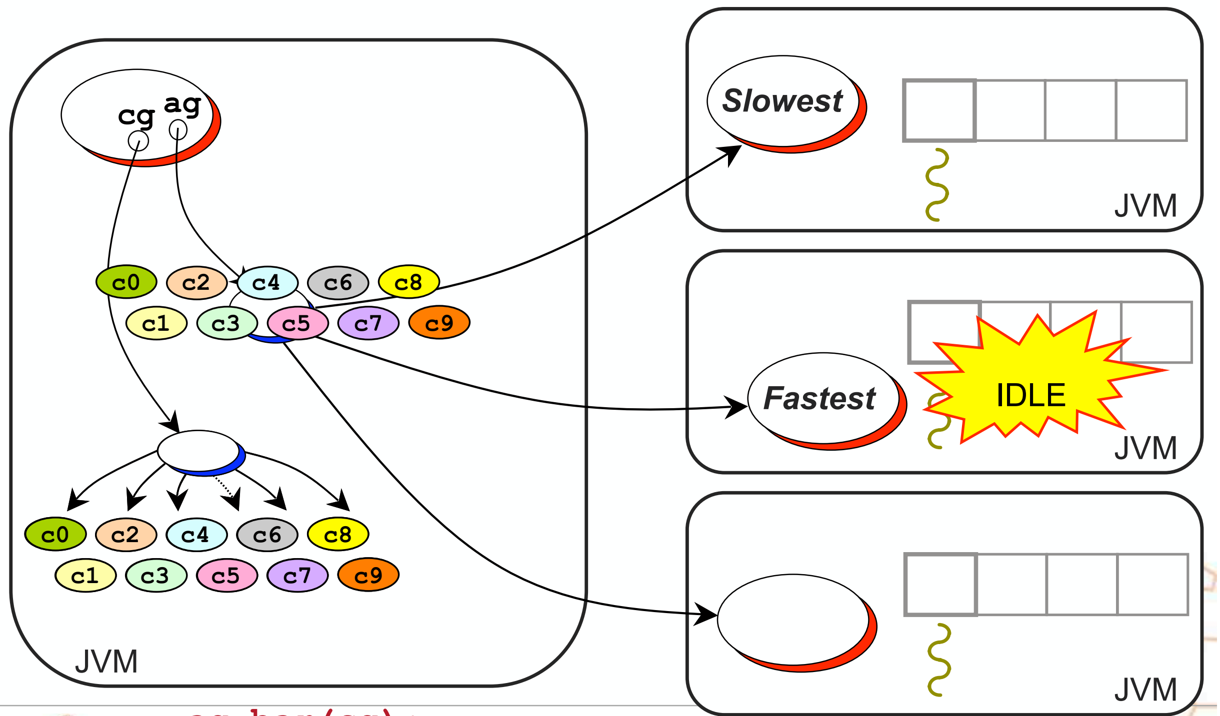
  ☺ Parameterized interactions

  ⇨ High-level programming **facilities**

Collective interfaces

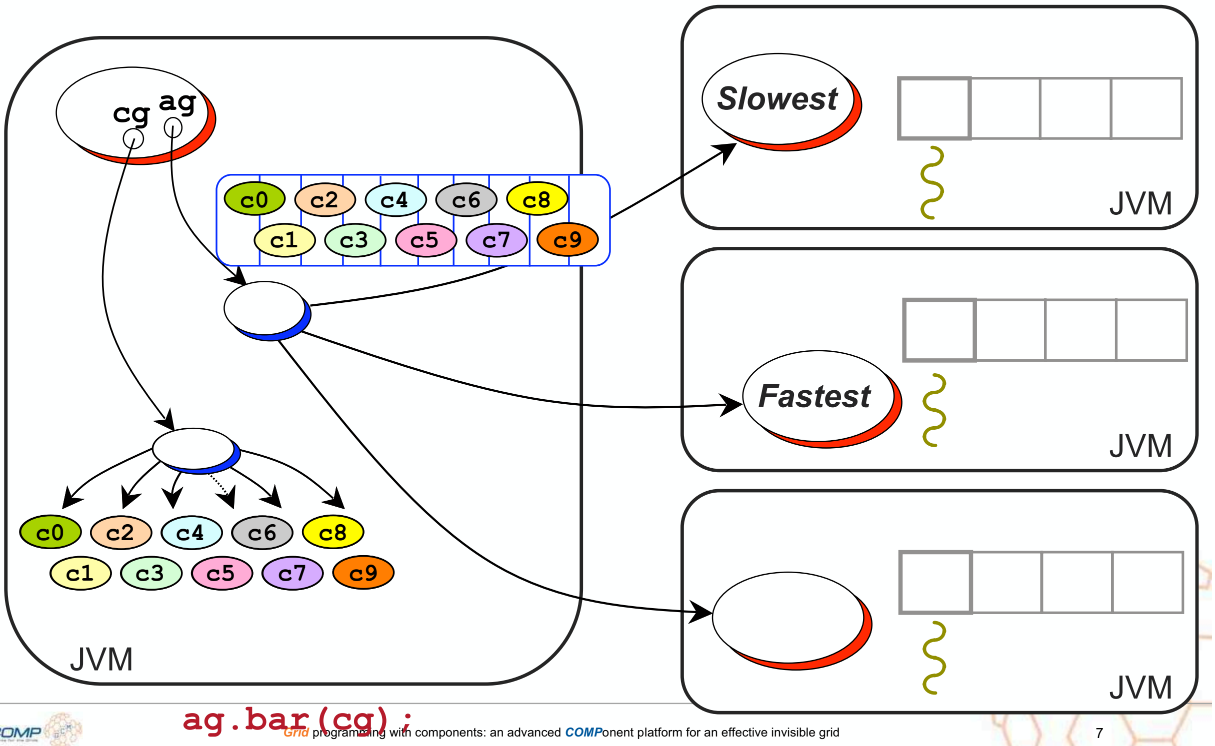


Static Dispatch Group

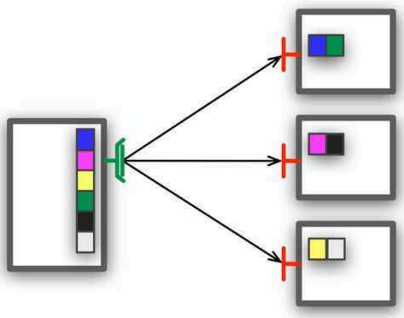


ag.bar(cg);

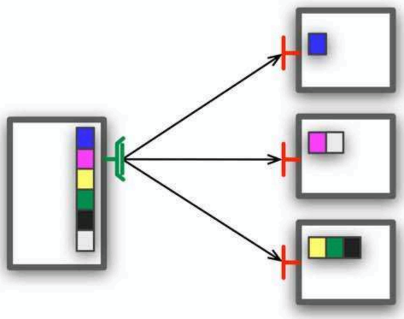
Dynamic Dispatch Group



Dynamic Dispatch with Multicast Interfaces



uniform distribution



knowledge-based distribution

more efficient !

Principles

- Minimal scheduling facilities
  - Knowledge-based scheduling
    workload **+** network congestion
- GCM programming model
- Composition oriented vs task oriented
- Low-level integration in ProActive/GCM

First achievements

- Load balances work units
- Compatible with POJO groups

- vs other frameworks:
  - Faster than ProActive's master-worker (low level)
  - Faster than *ourgrid* scheduler (fine grained tasks)
  - Comprehensive: splitting - scheduling - reduction
    (map-reduce / split-aggregate)

Impact on ProActive/GCM

- API preserved
- Extensions to Meta-Object Protocol
- Open implementation
- Integration to codebase: ProActive v4.0?

Side Contributions

- Bug fix for groups
  - “swallowed parameters” error : not all parameters distributed in some cases
- Relies on Java 5 concurrency features
  - More stable thus efficient for high loads

Future Work

- Finish integration (includes configuration spec)
- Use runtime load information
  - Aldinucci's work : tagging futures
- More standard dispatch modes
  - Random
  - Predictive CPU based?
- Unicast dispatch (probably short-term task)

Questions?

- applicability to adaptable farms?
  - ⇨ Parameterizable dispatch function
- suitability for GridCOMP use cases?
  - ⇨ Yes : simple mechanism

Contact:

Matthieu Morel mmorel@dcc.uchile.cl

Scheduling ProActive Applications using Gridbus Broker

Xingchen Chu, Srikumar Venugopal and Rajkumar Buyya

Grid Computing and **D**istributed **S**ystems (GRIDS) Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
www.gridbus.org

Gridbus Sponsors

Outline

- Objectives
- ProActive Scheduler Overview
- Gridbus Broker Overview
- ProActive and Gridbus Integration
- Remarks
- Conclusion and Future Work

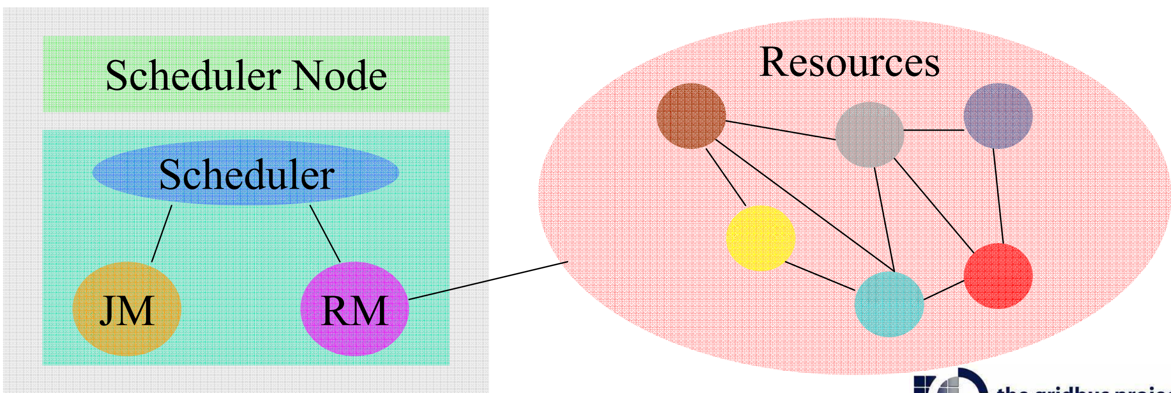
Objectives

- To make ProActive use the Gridbus Broker scheduling infrastructure
  - ProActive is able to leverage the economy-based and data-intensive scheduling algorithms provided by the Gridbus Broker
  - ProActive provides a programming environment for the Gridbus Broker

the gridbus project

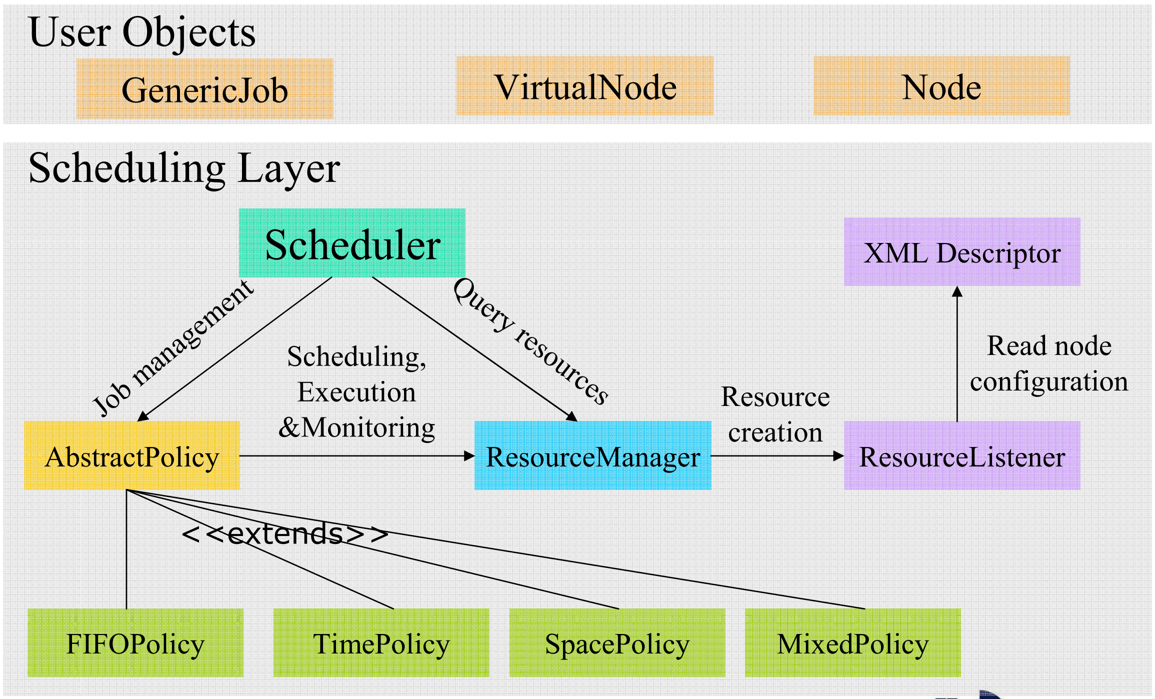
ProActive Scheduler Overview

- Scheduler is an active object itself
  - Talks to two objects: resource manager and job manager
  - Client needs to connect to the scheduler before running the application
- Resource manager allocated physical nodes to active objects
  - Resources are described using the XML descriptor
- Job Manager keeps track of status of active objects



the gridbus project

ProActive Scheduling Layer and Its dependencies



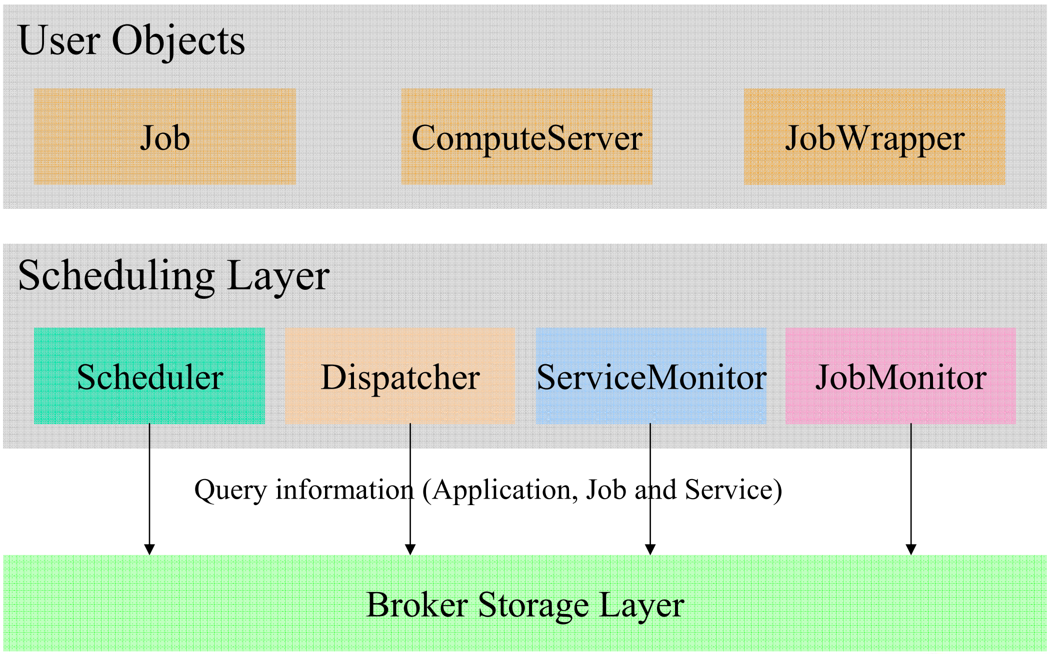
the gridbus project

Gridbus Broker Scheduling Overview

- Four main components
  - Scheduler, JobMonitor, ServiceMonitor and Dispatcher
  - Read and updates information via the Broker Storage layer
  - Various scheduling algorithms implemented such as round-robin, cost/time optimisation based on QoS parameter.

the gridbus project

Gridbus Broker Scheduling



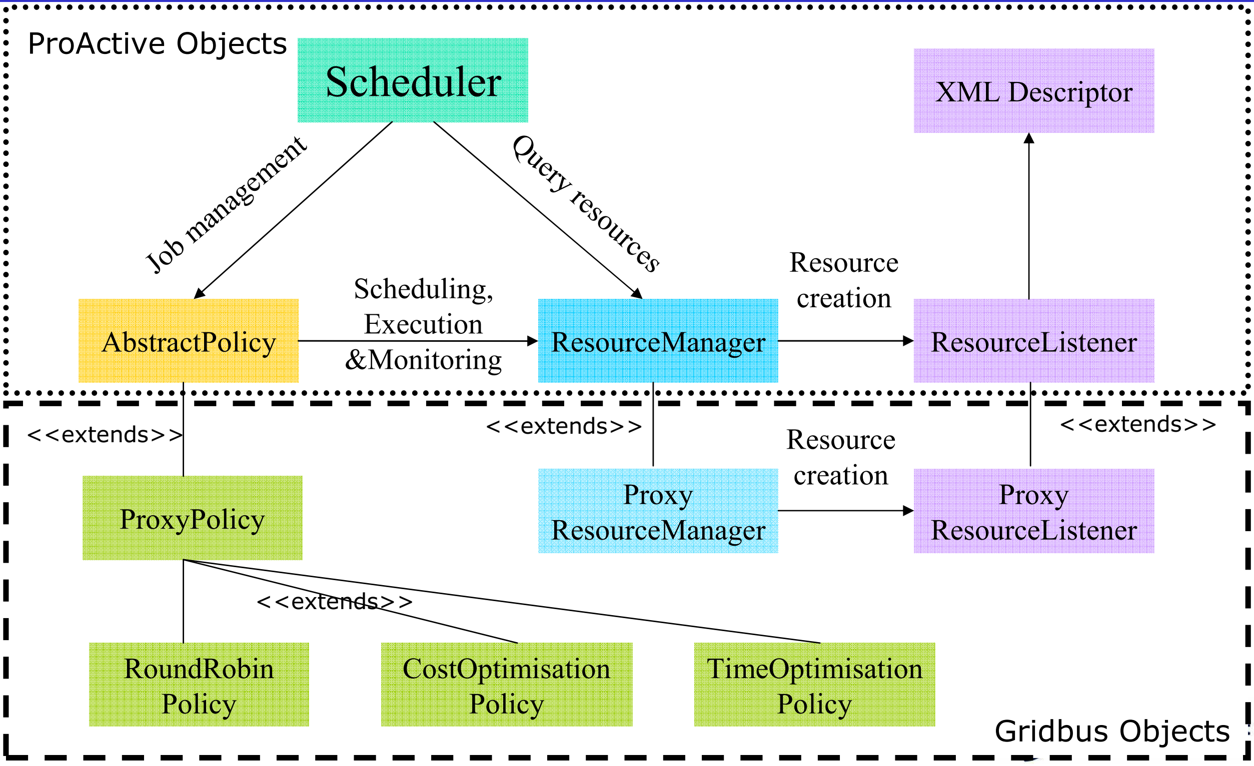
the gridbus project

ProActive Gridbus Integration : Challenges

- Minimise the impact on both systems
  - Each complex system should have as little knowledge of each other as possible
  - Each complex system only need to worry about its own terms and conditions
- Maximise the reusability of the existing infrastructure and codebase
  - Reuse the scheduling implementations provided by the Gridbus Broker
  - Reuse the ProActive runtime to execute applications
- Existing applications should work without changing the source code and recompiling

the gridbus project

ProActive Gridbus Integration (1)



ProActive Gridbus Integration (2)

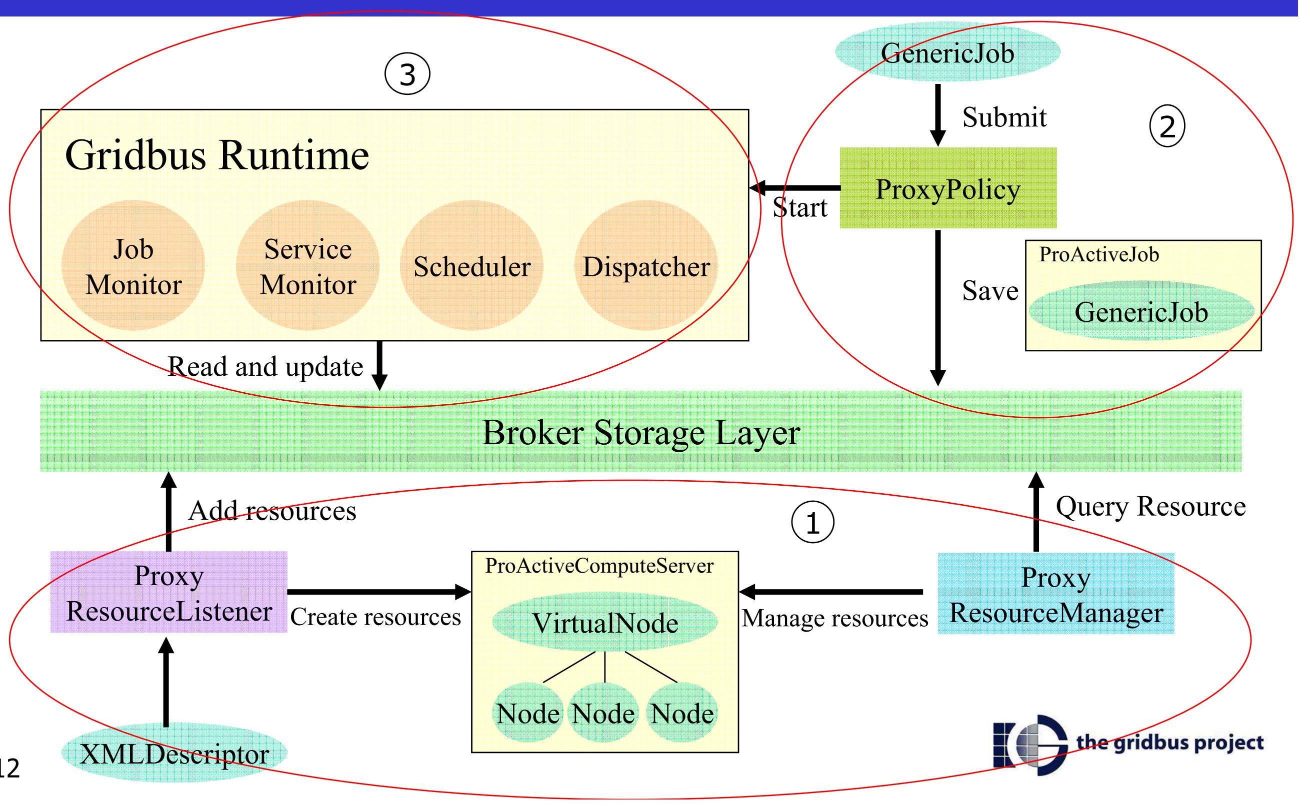
- ProxyPolicy
  - Initialises the Broker runtime services including the JobMonitor, ServiceMonitor, Scheduler, and Dispatcher
  - Implements job management functionality
  - Subclasses provide the information for the Broker to match the scheduling algorithms
- ProxyResourceManager
  - Overrides the ProActive's resource management functions using Broker's storage service.
- ProxyResourceListener
  - Listen for the resource creation by ProActive (virtual node, node)
  - Add resources into the broker storage system (compute server)

ProActive Gridbus Integration (3)

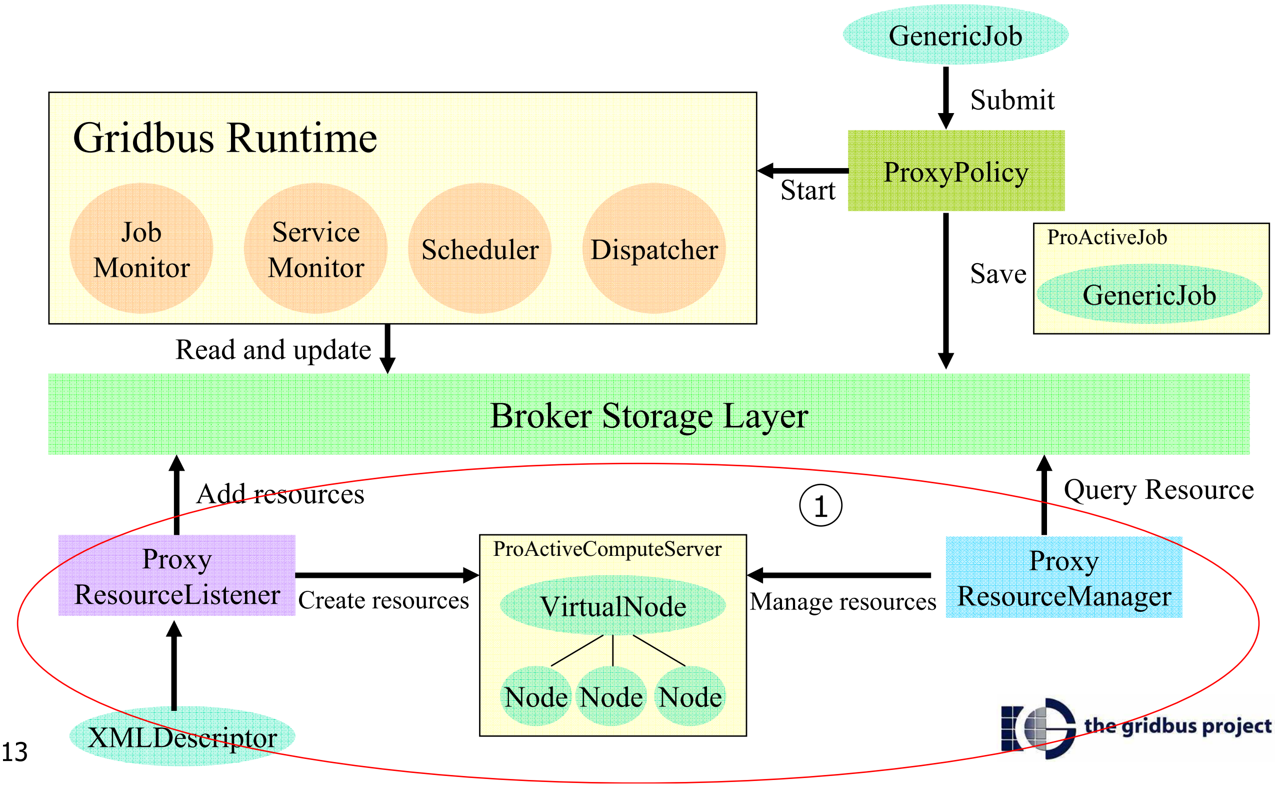
- ProActive user objects
  - Represent ProActive entities (GenericJob, Virtual Node, Node) in the Gridbus broker
- Entity Mapping
  - GenericJob -> ProActiveJob (extends Job)
  - Virtual Node, Node -> ProActiveComputeServer (extends ComputeServer)
  - ProActiveJobWrapper (extends the JobWrapper) for Broker to start active object

the gridbus project

Assemble the System

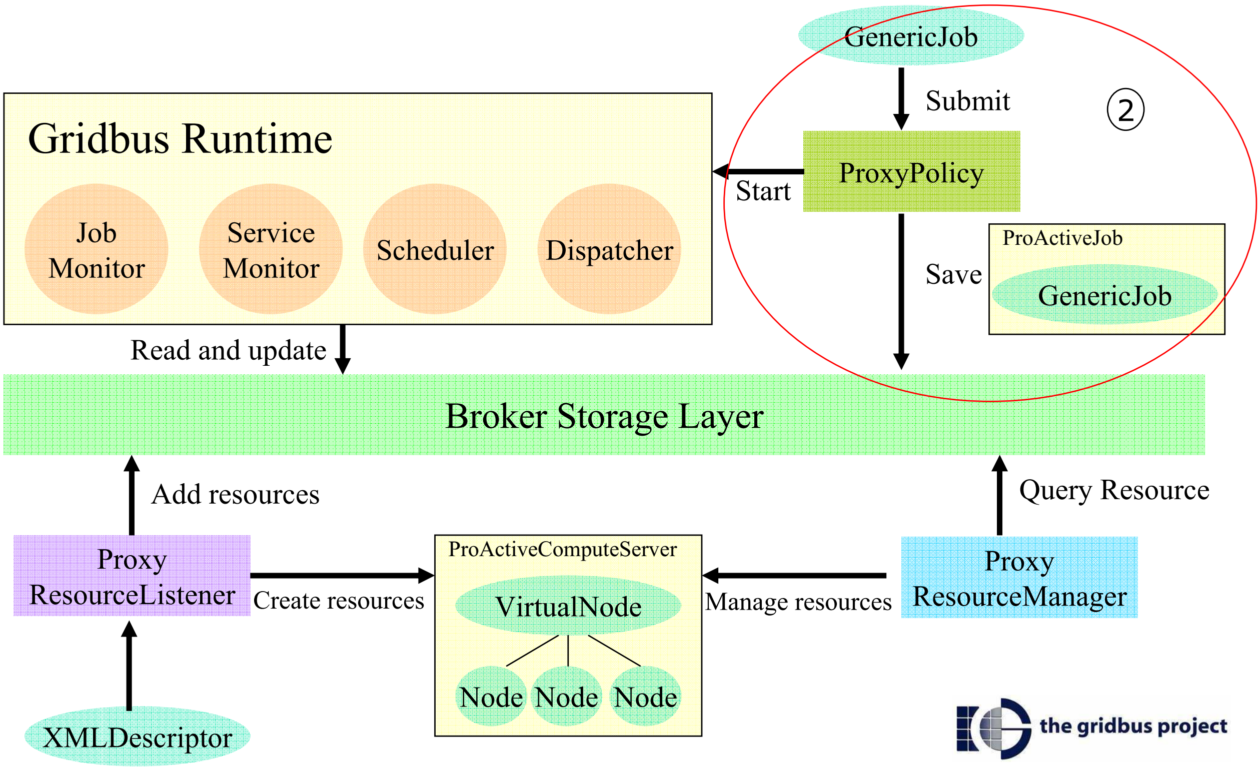


the gridbus project



Resource Acquisition and Management

- Proxy resource manager initialises the proxy resource listener to acquire the resources from the XML deployment descriptor
- The listener creates the ProActiveComputeServer objects based on the virtual node and nodes deployed by the XML
- The compute server objects are recorded into the Broker storage system
- The proxy resource manager simply manages those resources via the Broker storage system

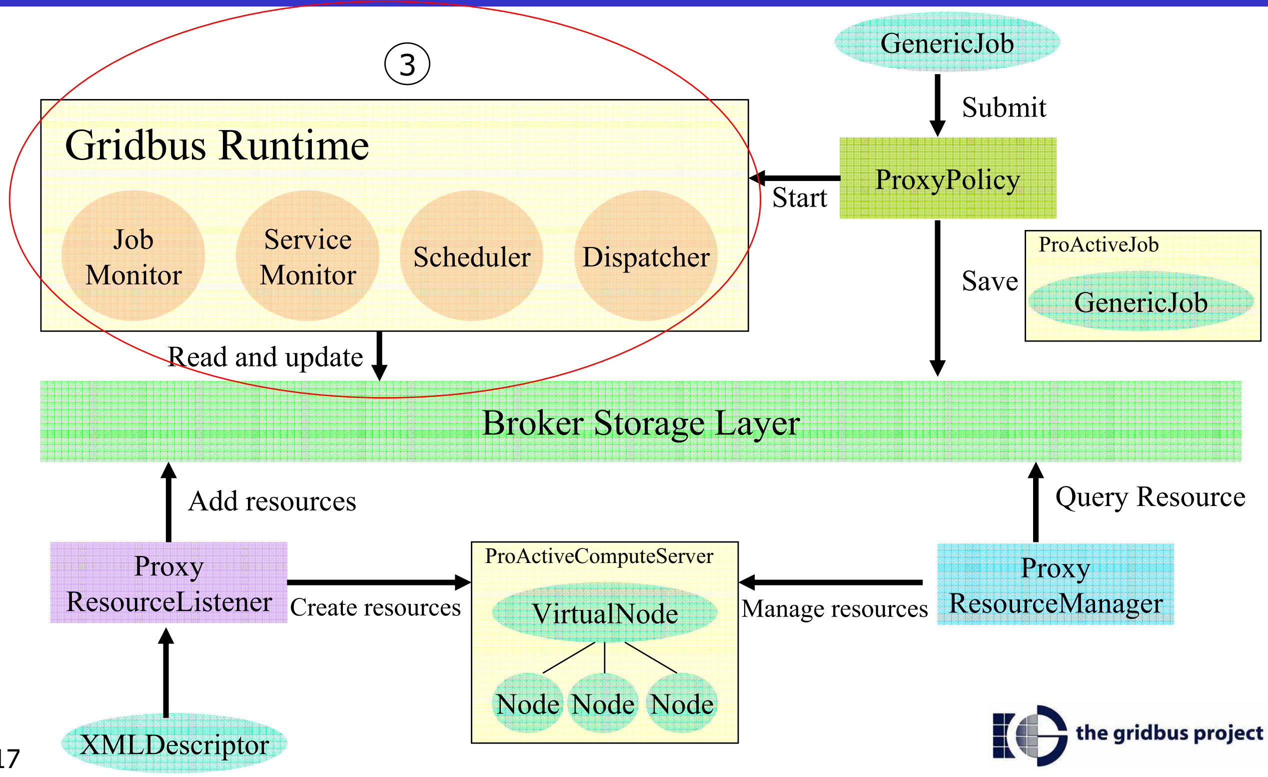


the gridbus project

Job Submission

- The ProActive user submit their applications in terms of GenericJob objects to the ProActive scheduler just as the same as before
- The proxy policy class creates the ProActiveJob object to wrap each GenericJob object
- The ProActiveJob objects are recorded into the Broker storage system for later schedule.

the gridbus project



Job Scheduling, Monitoring and Execution

- Fully controlled by the Broker scheduling infrastructure and totally transparent to the ProActive system
- The proxy policy class intialises the Broker runtime services as separate threads before any job submissions
- The runtime services periodically poll the storage system and retrieve or update required information.

the gridbus project

Validate the Solution

- Existing applications developed using ProActive should work without changing the source code and recompile.
- The dependencies between each system should be minimized.
  - Users from ProActive should be able to dynamically choose which scheduler to use either the existing one or the Gridbus broker's scheduler.
  - Gridbus broker should not be aware of any ProActive runtime information.
- Reuse the existing infrastructure at both side.
  - Job scheduling should be delegated to the Gridbus broker
  - ProActive's execution environment needs to be reused.

Run C3D render Application

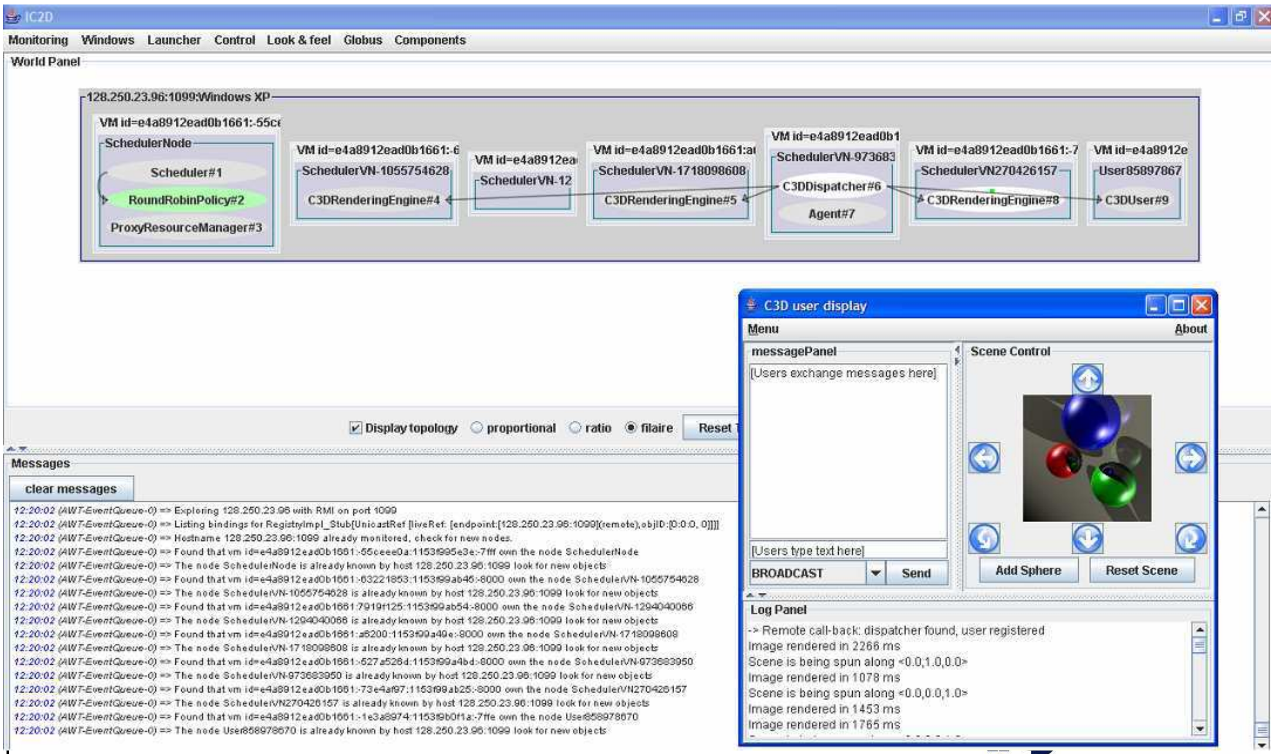
- Start the scheduler

```
public class StartScheduler{
   public static void main(String [] args){
     String policyName =
           "org.objectweb.proactive.scheduler.gridbus.policy.RoundRobinPolicy";
     String resourceManager =
           "org.objectweb.proactive.scheduler.gridbus.ProxyResourceManager";
     Scheduler.start(policyName,resourceManager);
   }
}
```

- Run the C3DRender

```
public class LaunchC3DRender {
    public static void main(String[] args) throws Exception {
        String schedulerURL = "rmi://localhost/SchedulerNode";
        Scheduler scheduler = Scheduler.connectTo(schedulerURL);
        String XML_LOCATION_UNIX =
             LaunchHello.class.getResource(
"/org/objectweb/proactive/examples/scheduler/c3d-render.xml")
    .getPath();
        scheduler.fetchJobDescription(XML_LOCATION_UNIX);
    }
}
```

C3D Render with IC2D



Remarks : On ProActive Side

- Minimum changes to the ProActive codebase
  - Only the scheduler class has been changed to add a new method supporting dynamic creation of proxy resource manager and proxy policy
- Client applications are unaware of the Gridbus Broker
- Existing ProActive runtime environment is reused
- Avoid classpath explosion
  - Only one extra jar-file is required: the Gridbus broker runtime library

Remarks: On Gridbus Side

- At the Gridbus Broker side
  - ProActive is just another type of middleware
  - Except the wrappers' implementations, nothing need to be modified, the broker is totally unaware of the existence of ProActive
  - Schedule and dispatch jobs as normal ones without worrying about the terms defined in ProActive such as GenericJob, Virtual Node and Node.

Conclusion and Future Work

- Provide an integration solution for ProActive using the Gridbus Broker for scheduling applications
  - Two complex systems are seamlessly working together without knowing each other
  - Existing ProActive examples can still work without recompiling.
  - The glue between the two system has been provided via the configuration file which is loaded dynamically at runtime through the existing deployment service
- Future Work
  - Closer integration of ProActive with economy-based scheduling
    - The job model of the ProActive has to be extended to support QoS parameters such as budget and deadline via a configurable way
  - Current implementation only focus on the RMI runtime provided by ProActive, a much more comprehensive testing needs to be done with other types of runtime environments provided by ProActive

Thanks for your attention!

Questions?

the gridbus project

Running Hello World

- Start the normal ProActive scheduler with two extra arguments
  - Class name of the ProxyResourceManager
  - Class name of the ProxyPolicy
- Run the LaunchHello program without extra configuration

```
public class StartScheduler{
   public static void main(String [] args){
     String policyName =
           "org.objectweb.proactive.scheduler.gridbus.policy.RoundRobinPolicy";
     String resourceManager =
           "org.objectweb.proactive.scheduler.gridbus.ProxyResourceManager";
     Scheduler.start(policyName,resourceManager);
   }
}
```

the gridbus project

LaunchHello In ProActive

```
public class LaunchHello {
    public static void main(String[] args) throws Exception {
        String schedulerURL = "rmi://localhost/SchedulerNode";
        Scheduler scheduler = Scheduler.connectTo(schedulerURL);
        String XML_LOCATION_UNIX =
             LaunchHello.class.getResource(
"/org/objectweb/proactive/examples/scheduler/job_template.xml")
   .getPath();
        scheduler.fetchJobDescription(XML_LOCATION_UNIX);
    }
}
```

Interoperability & cooperation between ProActive and XServices

Dr. ZHU Yan

zhuyanbuaa@hotmail.com

Leader of Web Services R. & D. Team
School of Computer Science & Engineering
Beihang University

About Us

- Web Services R&D Team in ACT (Institute of Advanced Computing Technology), School of Computer Science and Engineering, Beihang University
- ACT Members: *Currently 140+ researchers & developers*
  - Faculty: 3 Professors, 4 Associate Professors, 5 Lectures
  - Students: 48 PhD, 92 MS
- Web Services R. & D. Team is focusing on: Services platform and its applications
  - Service-Oriented Architecture & Enterprise Service Bus
  - Web Service Middleware and Platform
  - Web Service Workflow (E-Government, E-Commerce, etc.)
  - Web Service Portal (SMB, etc.)
  - Web Service Resource Framework (Sensor Network, etc.)
  - Web Service QoS
  - Web Service Cooperation (Seismic Analyses, Remote Sensing Satellite, etc.)
  - Multimedia Web Service (Remote Medical Treatment, ITS, etc.)
  - Semantic Web Service & Web 2.0 (Ajax)

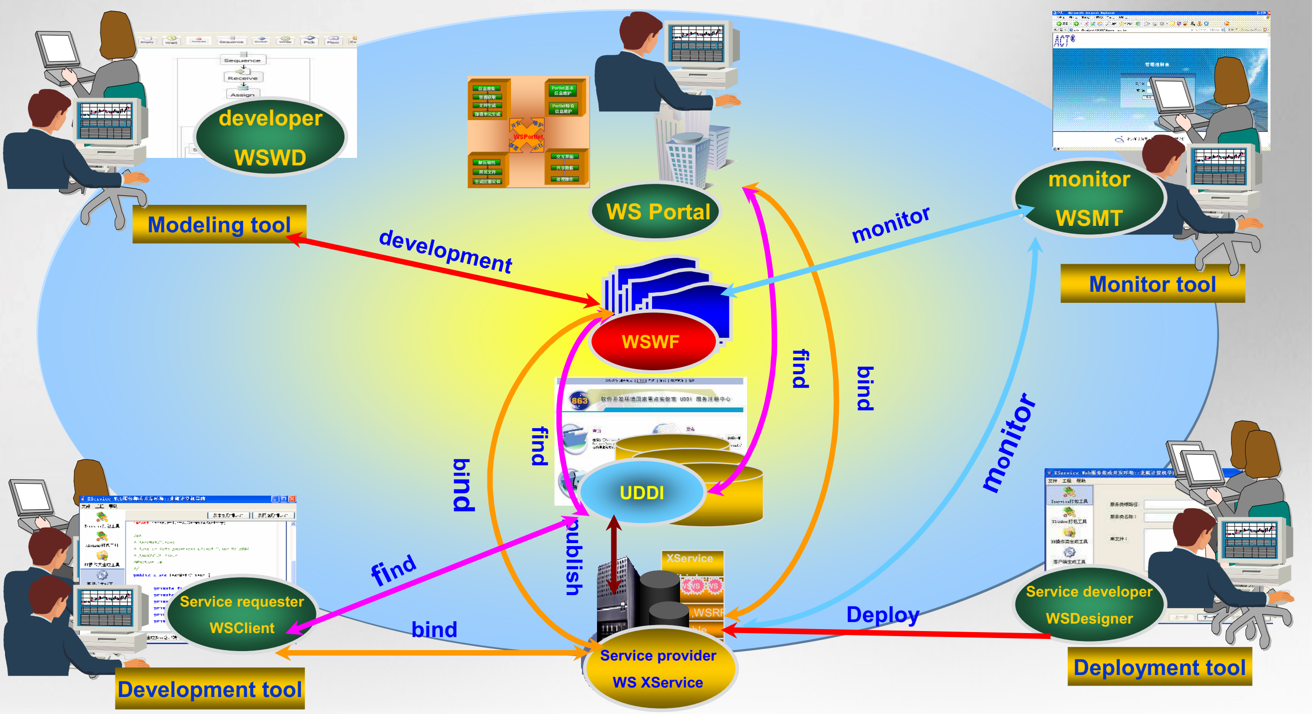
Research Background

- **XServices**: Web Services-based Application Supporting Environment
- **Motivation**
  - The trend: Web Service is a good way to build Internet-based Software.
  - Our target: To build a system environment for Web Service and Web Service based applications which can provide development assistance, deployment, runtime, monitoring, management for Web Service components and applications.

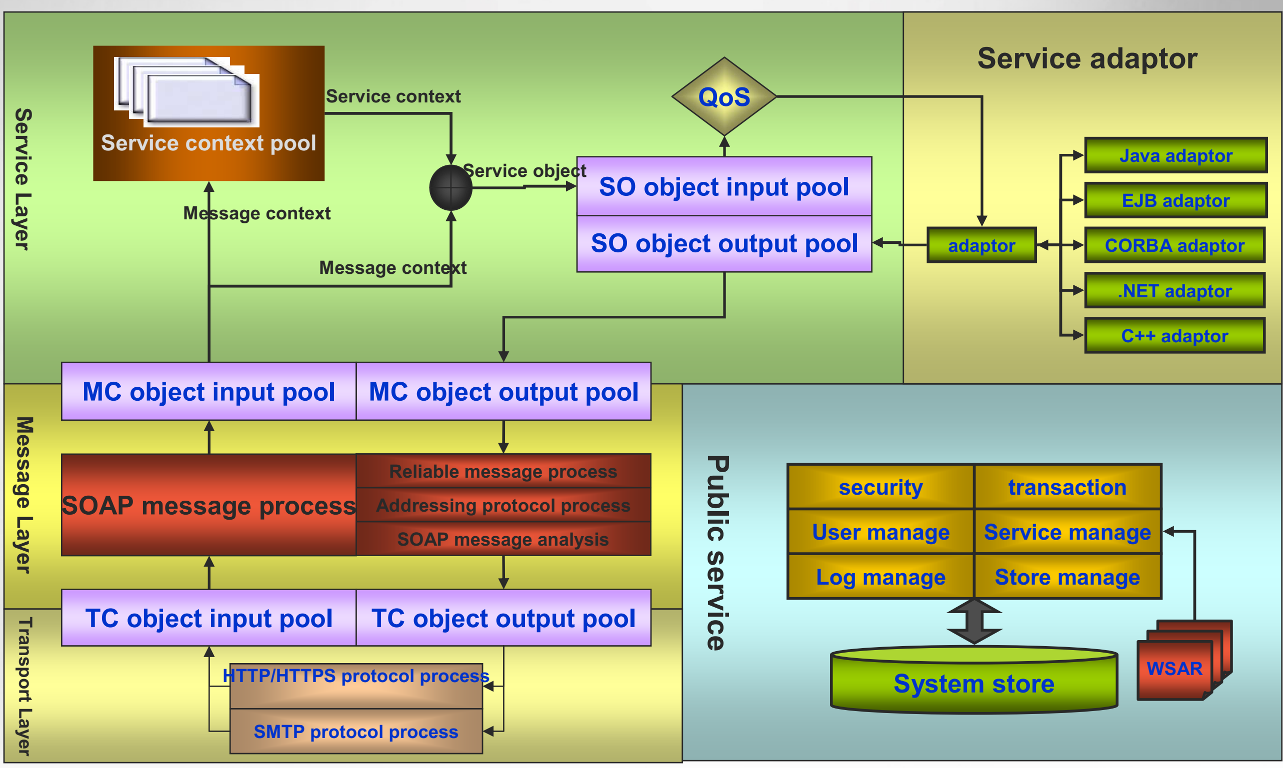
Research Background (const.)

- Funding Sources & Related Projects
  - A series of projects funded by National 863 Hi-tech Program and other Ministries (Over 10 million RMB)
    - Network Software Kernel Technologies and Runtime Platform, 2001
    - Web Service Transaction Middleware System, 2003
    - Web Service Information Platform, 2004
    - Web Service Software Technologies and Runtime Platform, 2004
    - Autonomic Computing and Service Collaboration Platform, 2006
    - ….
  - Application projects (Over 120 million RMB)
    - Web Services based E-Government Supporting Platform for Beijing City,2003
    - E-government Data Exchange Platform of Heilongjiang Province, 2004
    - Application Service Platform for United Productivity Information Co.,2004
    - CNGI (China Next Generation Internet) demonstration —— ITS demonstration ,2005
    - …

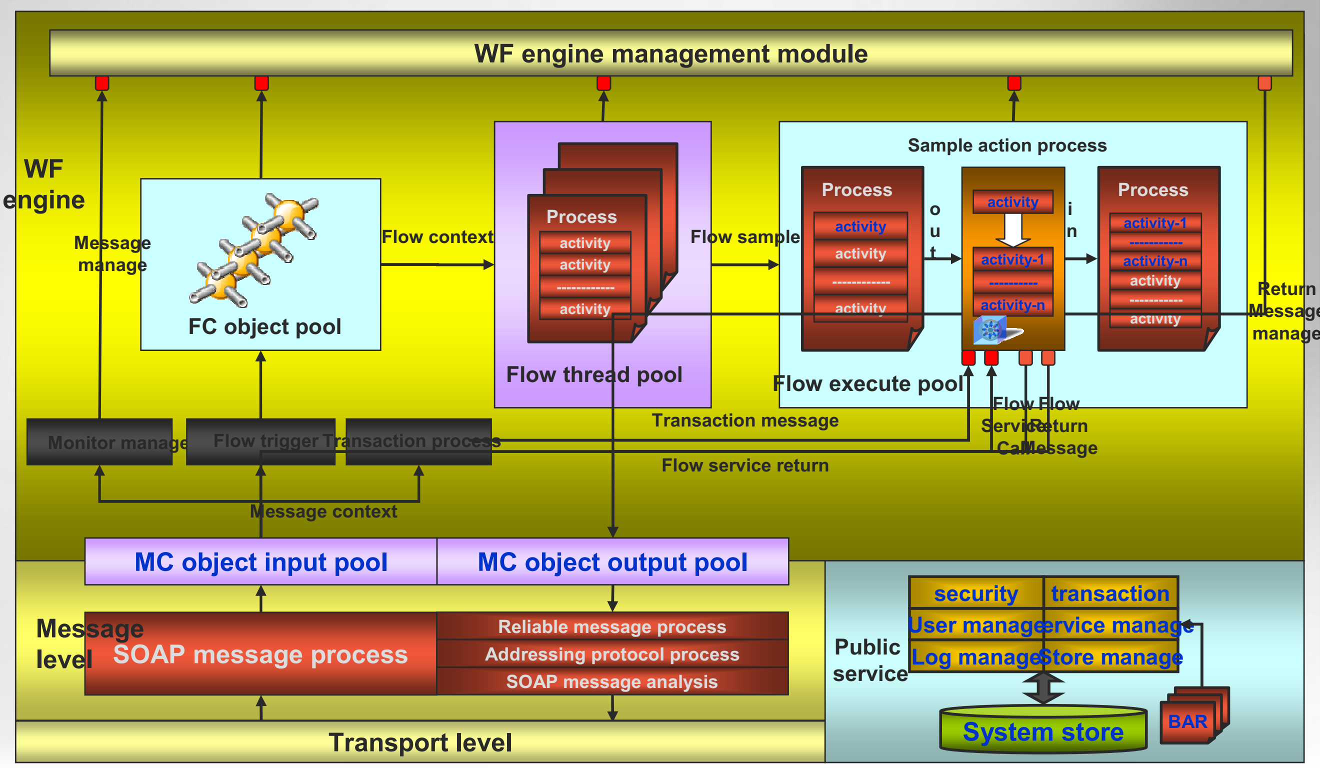
XServices: SOA Architecture Implementation



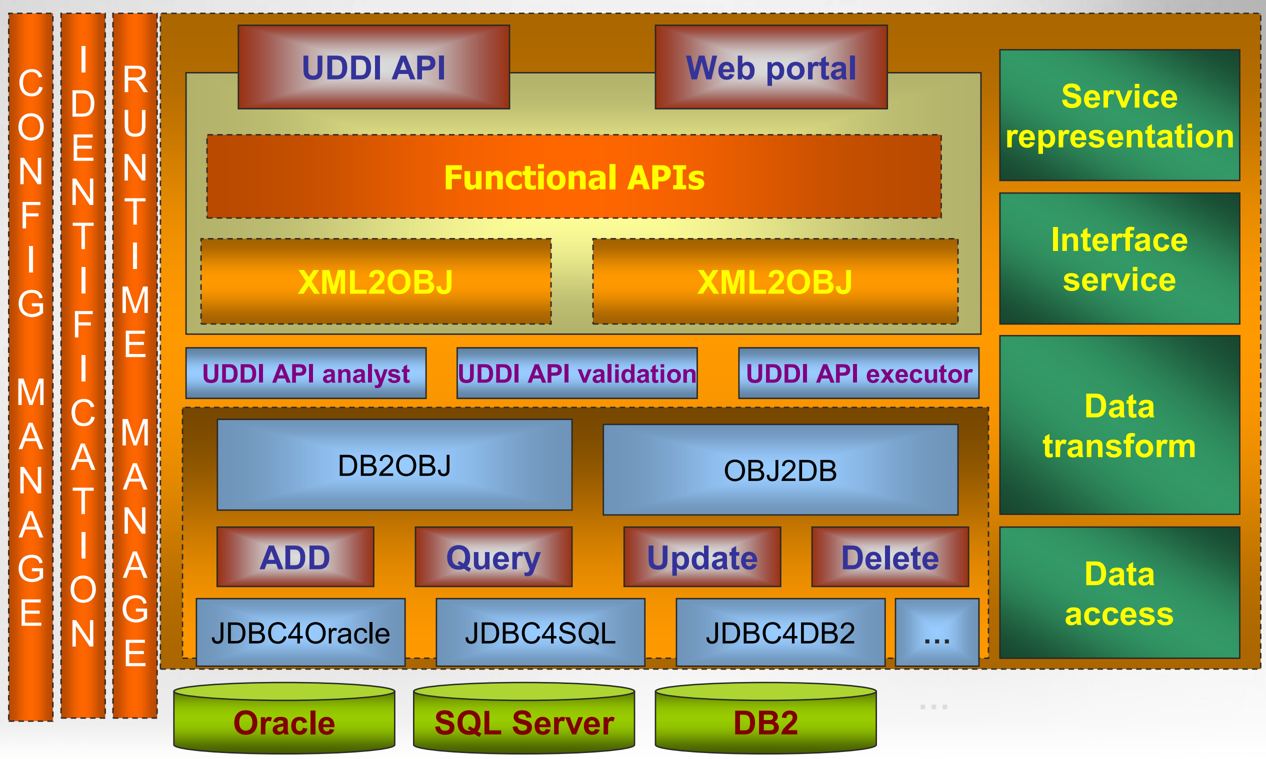
XRuntime: Web Services Application Server



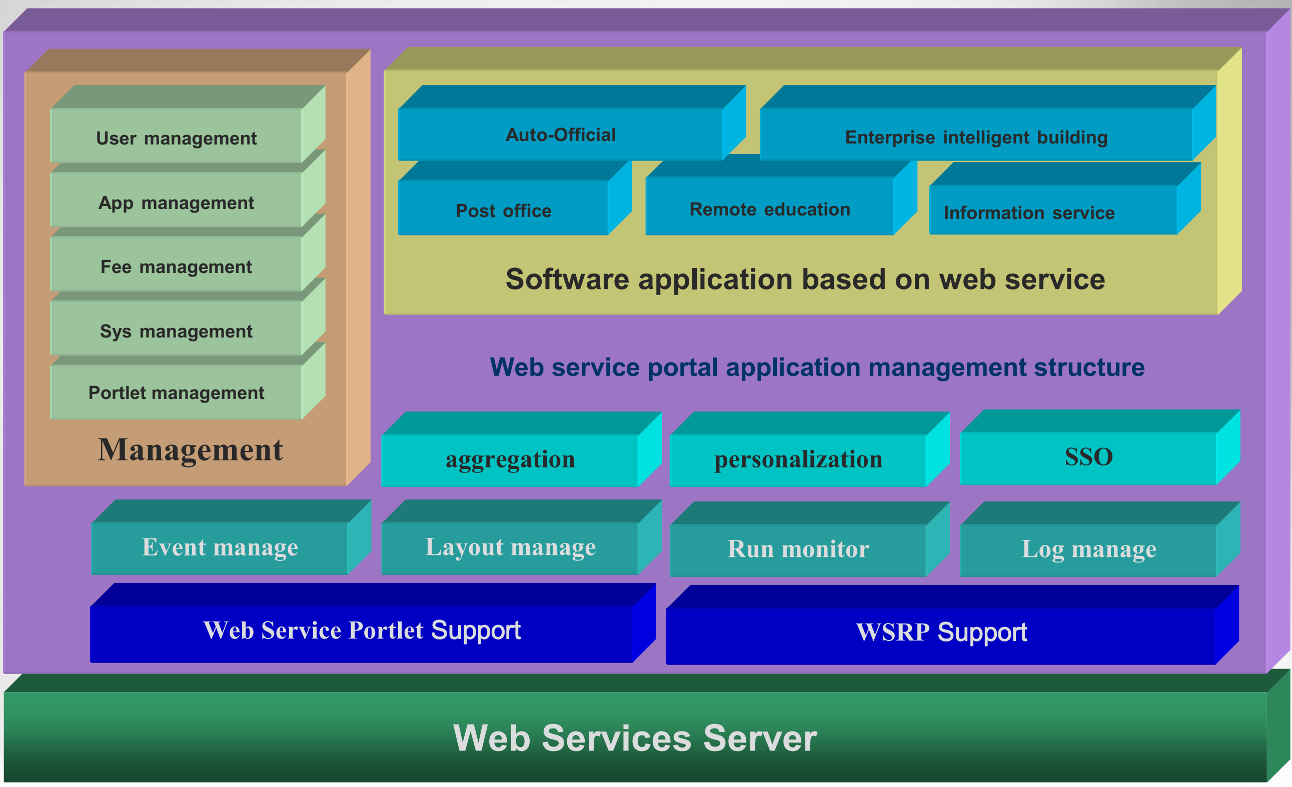
WSWF: Web Services-based Workflow



UDDI: Service Registry Center



WSPortal: Web Services-base Portal

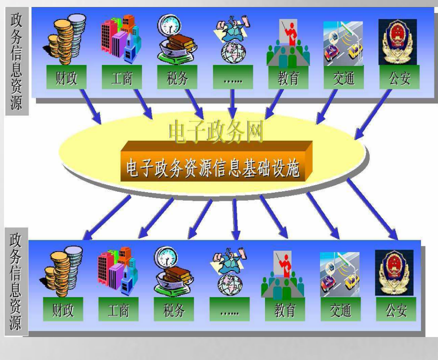


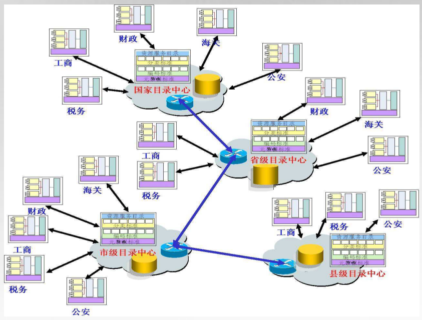
Our Achievements

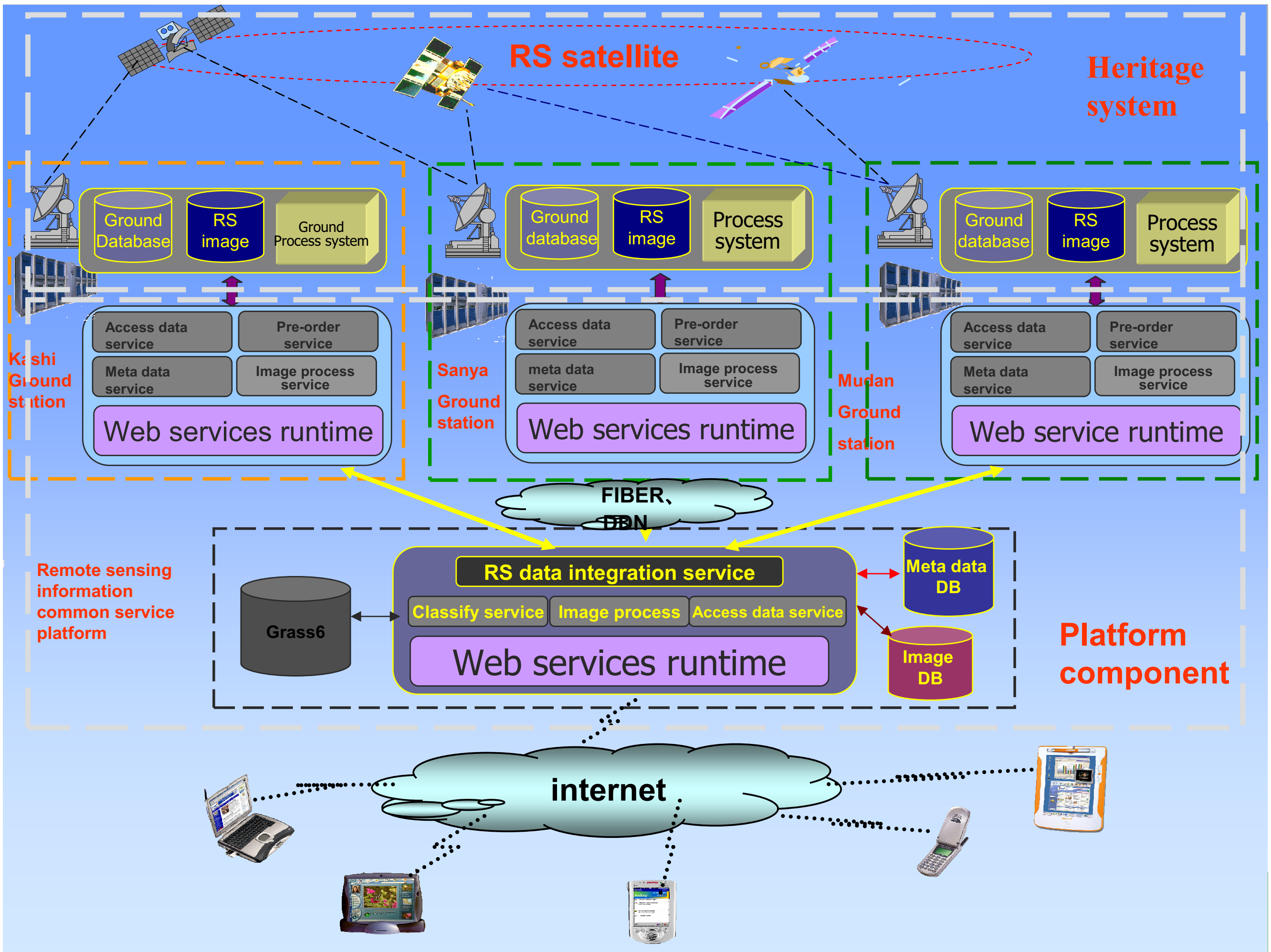
- **22 Software Copyrights Acquired**
  - Web Service Application Server （ No. 2005111824 ）
  - Web Service Application Supporting Environment System （ No. 2003SR7143 ）
  - UDDI Registry System （ No. 2003SR3015 ）
  - Web Service Running Management Console （ No. 2003SR7144 ）
  - Web Service Workflow Engine System （ No. 2003SR3016 ）
  - …
- **20 Software Patents Filed**
  - A Layered Web Service Handling Method (No.200510114783.7)
  - A Reliable Web Service Message Transportation Method (No.200510114566.8)
  - A BPEL Based Graphics to XML Documents Conversion Method (No.200510114689.1)
  - A Stack Based Web Service Workflow Handling Method (No.200510114563.4)
  - …

E-Government Application

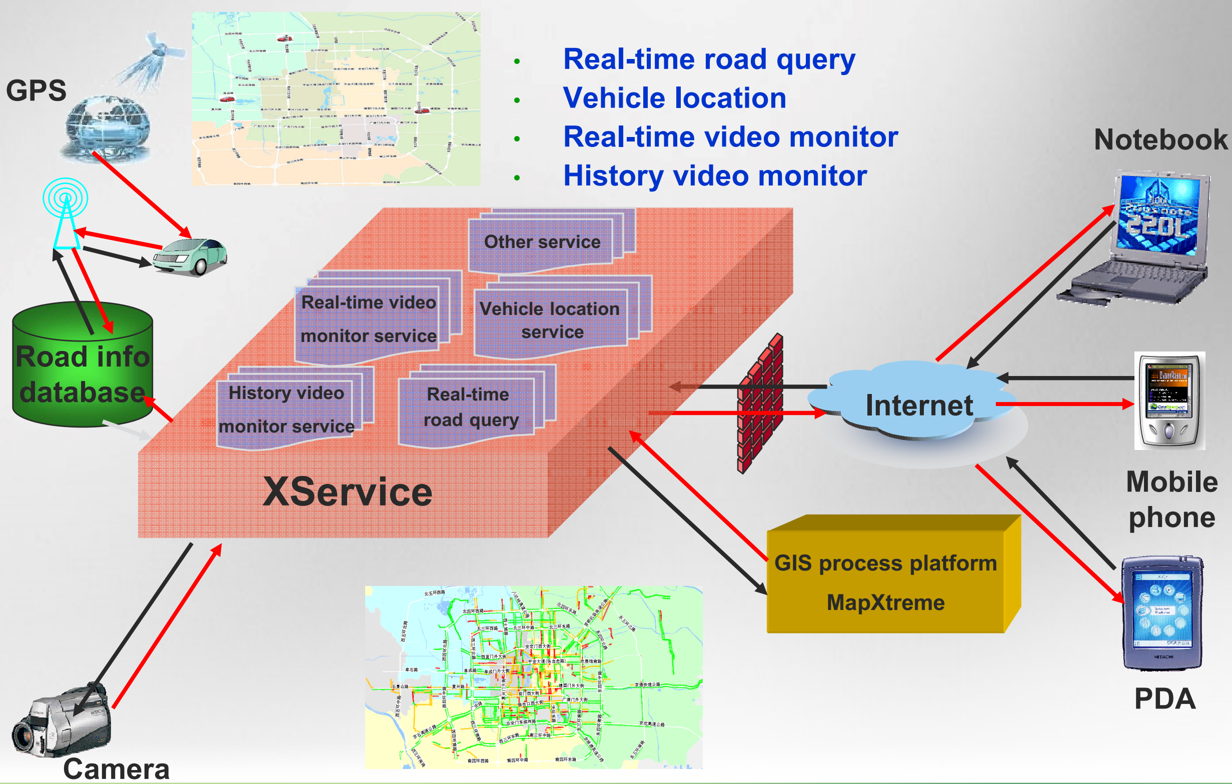
- **Background**
  - National E-Government Catalogue and Exchange Demo System
  - Some typical E-Government process were built according to the national standards. It is the first step to build E-Government systems widely.
- **Feature**
  - **Database Service Tools** is used to create the web services for E-Government information exchanging.
  - Web service created by departments is deployed in **XServices Runtime** in exchange environment to exchange information between departments.
  - E-Government process can be built quickly and visually by XService **Workflow** Designer and be examined by Debug function to build deployment packages of engine.







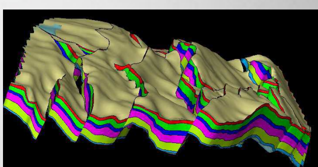
Intelligent Transportation System



Collaborative Visualization System of Seismic Model

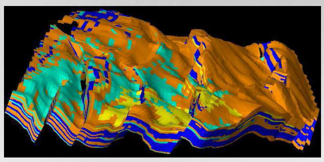
Background

- Geological researchers, Computer researchers and Seismic Analyzers in different location should process and discuss the same seismic model
- 3D seismic model is large scale(1000km^2 , 5GB)
- Necessary to visualize the model on mobile device



Key technology

- Remote visualization of large-scale 3d data （>1GB）
- Interoperability in heterogeneous platforms: client-server, server-server
- Collaboration between clients, Collaboration between services



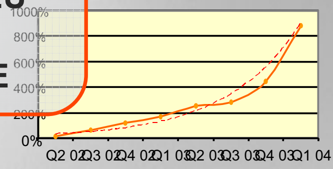
The OW2 Thrust

Community

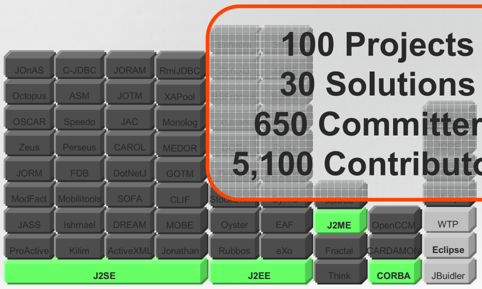
Size x 2 / year
60 Companies
2000 Individuals
80 Countries
250 Mailing-Lists
12,200 Suscribers

150,000 Visitors/m
2,400,000 Dwnlds '06
40% US, 40% EU
NESSI
ORIENTWARE

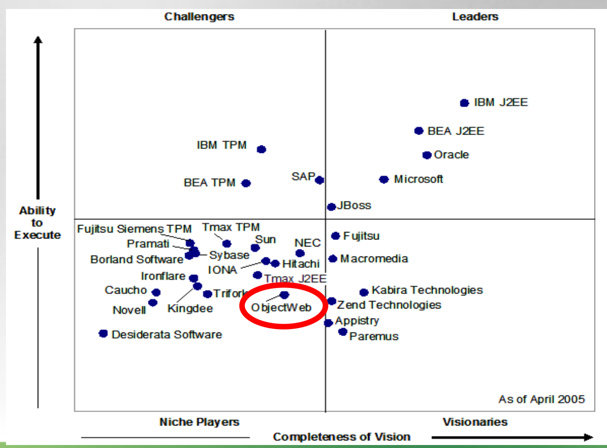
Impact



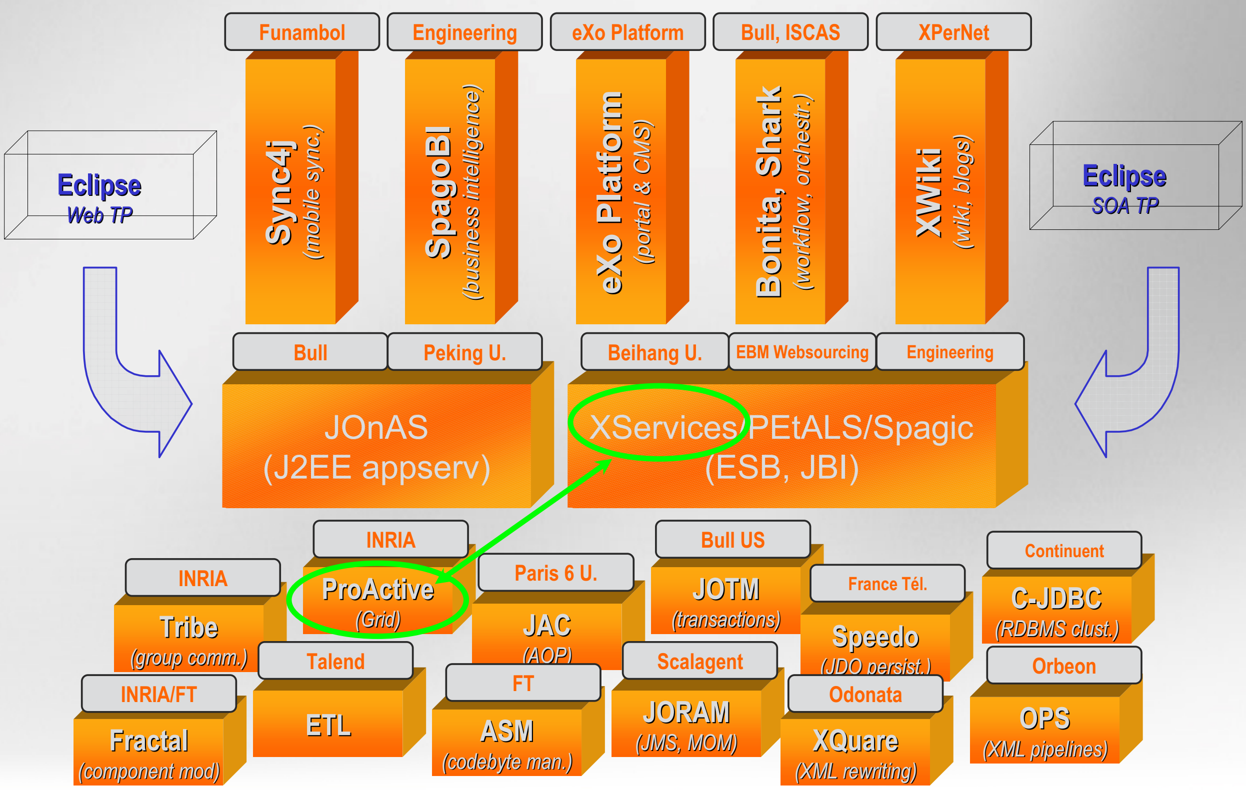
Software



100 Projects
30 Solutions
650 Committers
5,100 Contributors



OW2 Java Commitment



SOA Main Principles - Denis

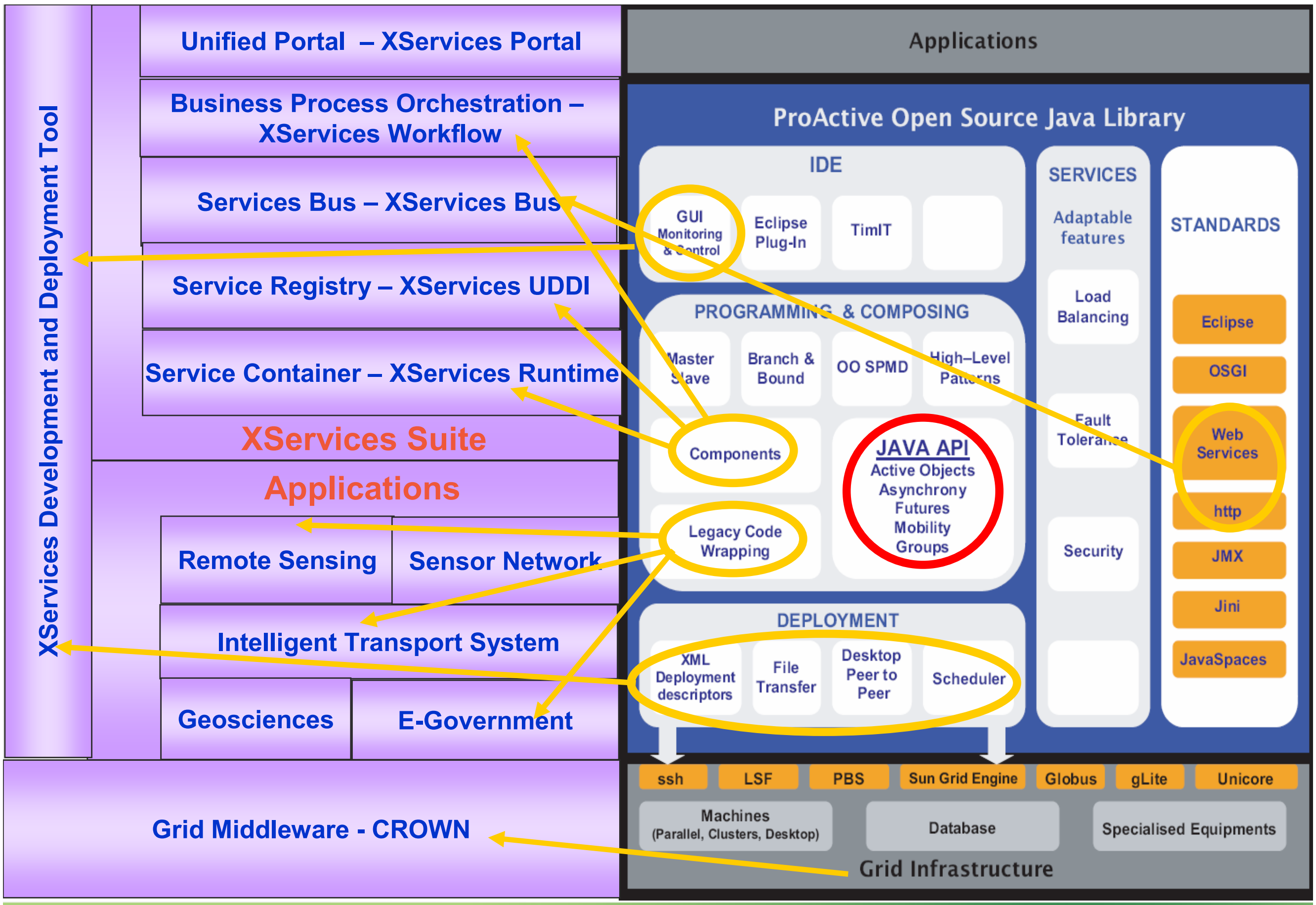
- service
- ibility, F
- ecture
- pplicati s
-
-
- sed on
- ent tec
- red by t
- wiring

SOA:

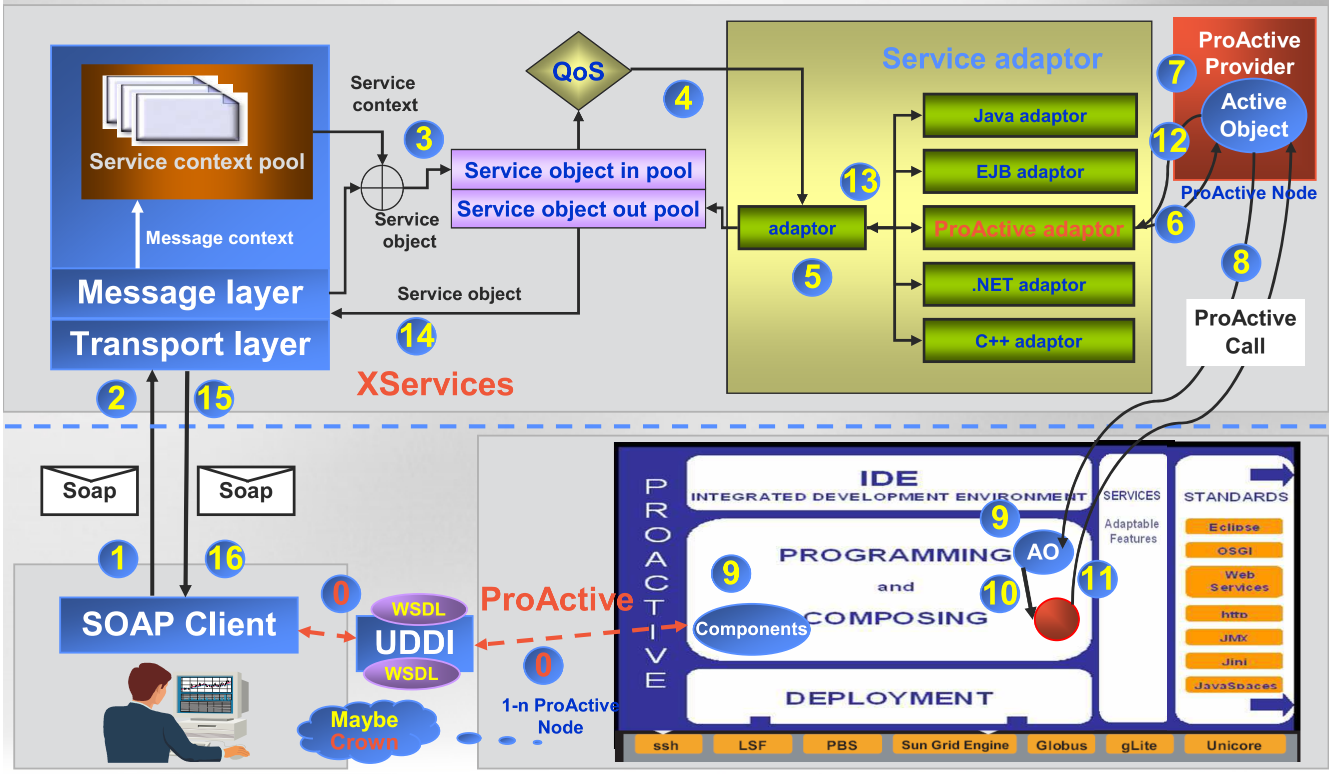
An architecture to Loosely Couple applications as services

ProActive:

A middleware (Core: Java API) to Program Coupled, //, Distributed, Multi-Threaded applications and seamlessly integrate in SOA



From XServices to ProActive



Merci

Grid programming with components:
an advanced ***COMP***onent platform
for an effective invisible grid

GridCOMP
Effective Components for the Grids

Session 3 - Perspectives & Panel

GridCOMP Workshop
October 31st, 2007
CNIC, Beijing, China

Generating Safe GCM Components

Antonio Cansado Eric Madelaine

INRIA Sophia Antipolis

GridCOMP - Pekin, 2007

INRIA

Motivation Prototype Diagrams for GCM Components Generation of Safe Components Conclusions

Outline

INRIA

Outline

Reusing and Assembling Components

- Safe Assembly of Components
  - Static typing of bound interfaces
  - Compatibility of dynamic behaviour
    - Formal specification of Components
- Choice
  - Integrate ADL and BDL
- Difficulty
  - Provide a framework for non-specialists

Reusing and Assembling Components

- Safe Assembly of Components
  - Static typing of bound interfaces
  - Compatibility of dynamic behaviour
    - Formal specification of Components
- Choice
  - Integrate ADL and BDL
- Difficulty
  - Provide a framework for non-specialists

Motivation Prototype Diagrams for GCM Components Generation of Safe Components Conclusions
The Need

Reusing and Assembling Components

- Safe Assembly of Components
  - Static typing of bound interfaces
  - Compatibility of dynamic behaviour
    - Formal specification of Components
- Choice
  - Integrate ADL and BDL
- Difficulty
  - Provide a framework for non-specialists

INRIA

Outline

Structure of the Framework

Specify using High-Level Specification Language
- Vercors Component Environment (VCE)
- UML 2

Generate behavioural models
- Validate and Verify

Generate Java control code
- Strong guarantees

Structure of the Framework

Specify using High-Level Specification Language
- Vercors Component Environment (VCE)
- UML 2

Generate behavioural models
- Validate and Verify

Generate Java control code
- Strong guarantees

Motivation | Prototype | Diagrams for GCM Components | Generation of Safe Components | Conclusions
Approach

Structure of the Framework

Specify using High-Level Specification Language
- Vercors Component Environment (VCE)
- UML 2

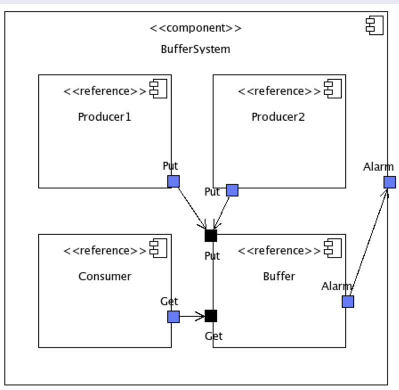
Generate behavioural models
- Validate and Verify

Generate Java control code
- Strong guarantees

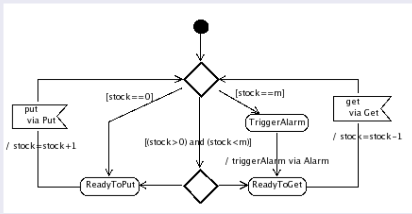
Outline

Unifying ADL and BDL

Component Diagram



State Machine Diagram

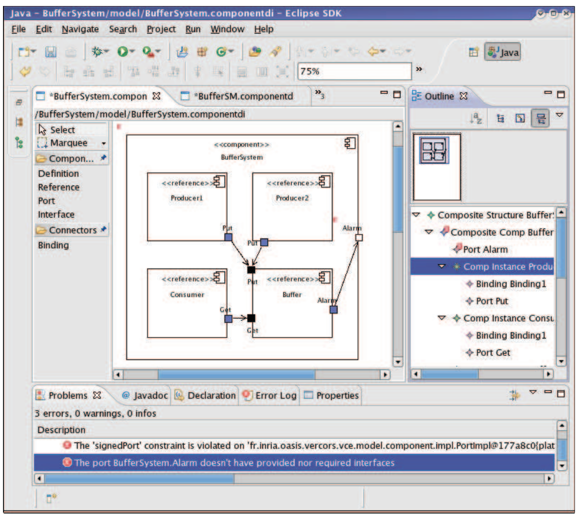


Data influencing the control-flow and the topology

Prototype

- Functional specification of components
- Component libraries
- *Bottom-up* and *Top-down* specification
  - Specification given as a State Machine
  - Implementation given as a composition of subcomponents
- Integrated into Eclipse as plugins
- Generation of behavioural model

Snapshot



Validate and Verify

- Sound semantic model – pNets
  - Hierarchical, Parameterized Networks of Labelled Transition Systems
- Generate Behavioural Models
  - Functional and Non-Functional concerns
- Model-checking
  - Deadlocks, Reachability, Safety, Liveness
  - Properties specified as automata
  - Functional and Non-Functional verification

Motivation Prototype Diagrams for GCM Components Generation of Safe Components Conclusions
Vercors Component Environment

Validate and Verify

- Sound semantic model – pNets
  - Hierarchical, Parameterized Networks of Labelled Transition Systems
- Generate Behavioural Models
  - Functional and Non-Functional concerns
- Model-checking
  - Deadlocks, Reachability, Safety, Liveness
  - Properties specified as automata
  - Functional and Non-Functional verification

Validate and Verify

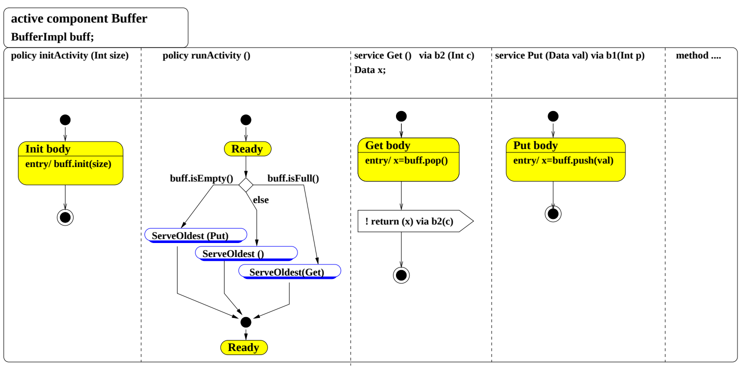
- Sound semantic model – pNets
  - Hierarchical, Parameterized Networks of Labelled Transition Systems
- Generate Behavioural Models
  - Functional and Non-Functional concerns
- Model-checking
  - Deadlocks, Reachability, Safety, Liveness
  - Properties specified as automata
  - Functional and Non-Functional verification

Outline

Missing Features

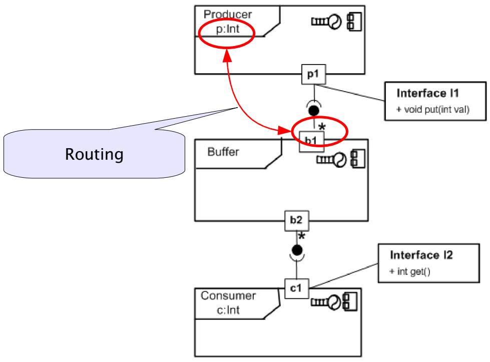
- Asynchronous components
  - Method calls performed on client interfaces → Future
  - Data-usage → Wait-by-necessity
- Collective interfaces
- Parameterized components
- NF management

Decomposing the Behaviour



Generate skeletons for GCM components

Parameterized Topologies



Code Generation

- Goal
  - Same behaviour as the specification
- Java code
  - GCM ADL
  - Final code of Fractal controllers
  - Skeletons of `runActivity()` and methods
- Hooks to fill-in final implementation
  - User-defined Business code

Code Generation

- Goal
  - Same behaviour as the specification
- Java code
  - GCM ADL
  - Final code of Fractal controllers
  - Skeletons of `runActivity()` and methods
- Hooks to fill-in final implementation
  - User-defined Business code

Motivation Prototype Diagrams for GCM Components Generation of Safe Components Conclusions

Code Generation

- Goal
  - Same behaviour as the specification
- Java code
  - GCM ADL
  - Final code of Fractal controllers
  - Skeletons of `runActivity()` and methods
- Hooks to fill-in final implementation
  - User-defined Business code

INRIA

Outline

Generate GCM / ProActive code

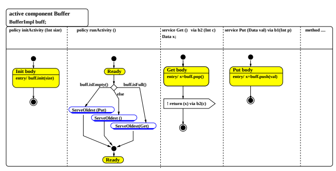
Fractal Controllers

```
public String[] listFc() {
  return new String[]{ CASHBOXEVENTIF_BINDING };
}
public Object lookupFc (String clientItfName) {
  if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
    return cashBoxEventIf;
  return null;
}
public void bindFc (String clientItfName, Object serverItf) {
  if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
    cashBoxEventIf = (CashBoxEventIf)serverItf;
}
public void unbindFc (String clientItfName) {
  if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
    cashBoxEventIf = null;
}
```

Outline

Generation of Skeletons: From State Machines

- `runActivity()`
  - Service policy
- Service methods
  - Server Interfaces of Primitive Components
- Control-flow and data-flow
  - Control structure
  - Method calls performed on client interfaces
  - Data-usage points



Service Policy

runActivity()

```
public void runActivity(Body body) {
  Service service = new Service(body);
  while (body.isActive()) {
    cashBoxEventIf.saleStarted();
    cashBoxEventIf.saleFinished();
    if ((new AnyBool()).prob(50)) {
      cashMode();
      cashAmount();
      service.blockingServeOldest("changeAmountCalculated");
      cashBoxEventIf.cashBoxClosed();
    }else
      creditCardMode();
} }
```


Control-Flow and Data-Flow

Service Method

```
public void pinEntered(PIN pin) {
  if (creditInfo != null) {
    Transaction transId = bankIf.validateCard(creditInfo, pin);
    if (ProActive.getFutureValue(transId) != null) {
      Info info = bankIf.debitCard(transId, runningTotal);
      if (ProActive.getFutureValue(info) != null){
        Sale sale = new SaleImpl(
                  new PaymentModeImpl(PaymentModeImpl.CREDIT),
                  products, runningTotal);
                  saleRegisteredIf.bookSale(sale);
        info.getInfo(); // wait-by-necessity
        init();
  ...
}
```

Conclusions and Perspectives

Short-term

- Tool for GCM Specification
- Validation of Behavioural properties
- Generation of Safe code

Long-term

- Multicast / Gathercast interfaces
- Specify Non-Functional controllers in the membrane

Motivation Prototype Diagrams for GCM Components Generation of Safe Components Conclusions

Conclusions and Perspectives

Short-term

- Tool for GCM Specification
- Validation of Behavioural properties
- Generation of Safe code

Long-term

- Multicast / Gathercast interfaces
- Specify Non-Functional controllers in the membrane

INRIA

References

http://www-sop.inria.fr/oasis/Vercors

S. Ahumada, L. Apvrille, T. Barros, A. Cansado, E. Madelaine, and E. Salageanu.
Specifying Fractal and GCM Components With UML.
In *Proc. of the XXVI International Conference of the Chilean Computer Science Society (SCCC'07)*, Iquique, Chile, November 2007. IEEE.

A. Cansado, D. Caromel, L. Henrio, E. Madelaine, M. Rivera, and E. Salageanu.
A Specification Language for Distributed Components implemented in GCM/ProActive.
Lecture Notes in Computer Science. Springer, (To be published) 2007.

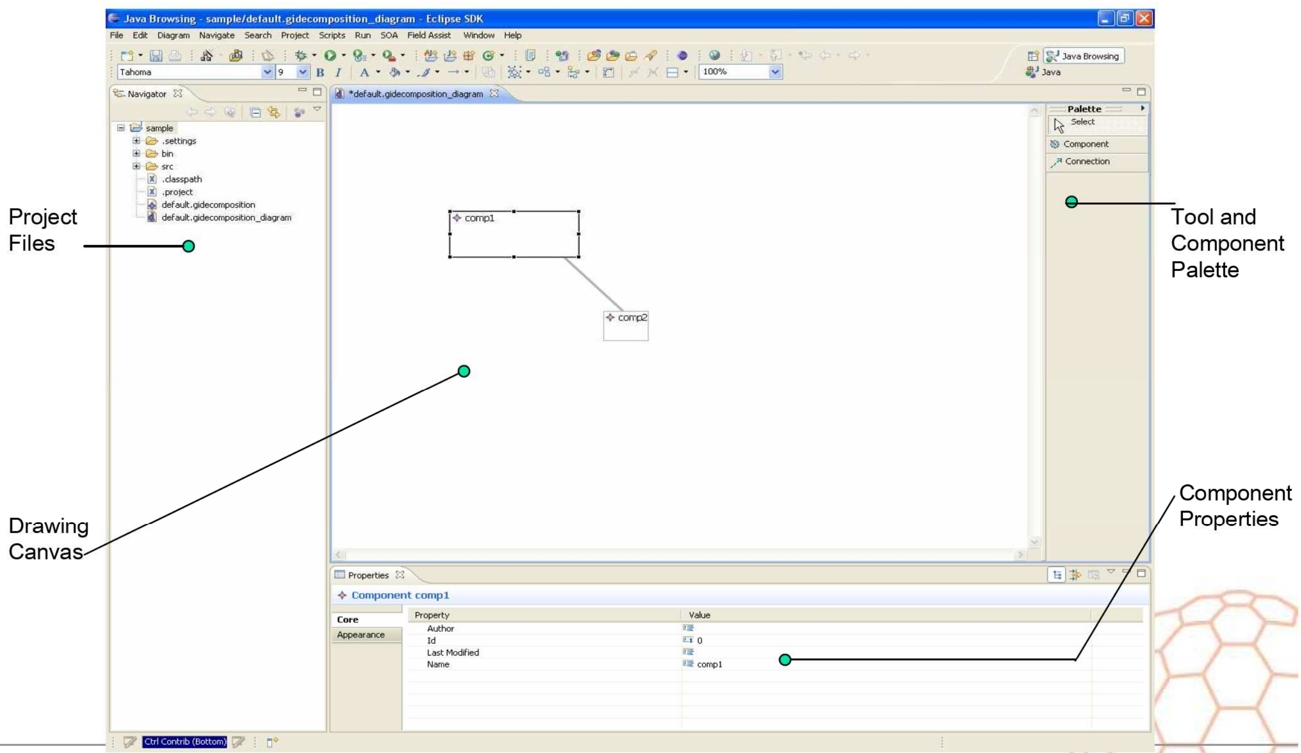
A. Cansado, L. Henrio, and E. Madelaine.
Towards real case component model-checking.
In *5th Fractal Workshop*, Nantes, France, July 2006.

GridCOMP
Effective Components for the Grids

Grid IDE

Artie Basukoski
University of Westminster

User Interface of the Composition Editor



Domain model

```
<?xml version="1.0" encoding="utf-8" ?>
- <xsd:schema targetNamespace="http://perun.hscs.wmin.ac.uk/GridCOMP/gidecomposition"
    xmlns:gidecomposition="http://perun.hscs.wmin.ac.uk/GridCOMP/gidecomposition"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore">
    <xsd:element name="Component" type="gidecomposition:Component" />
  - <xsd:complexType name="Interface">
      <xsd:attribute name="Name" type="xsd:string" use="required" />
      <xsd:attribute name="Id" type="xsd:unsignedShort" use="required" />
      <xsd:attribute name="Type" type="xsd:string" use="required" />
      <xsd:attribute name="Cardinality" type="gidecomposition:CardinalityType" use="required" />
      <xsd:attribute name="CardinalityIn" type="xsd:unsignedByte" use="required" />
      <xsd:attribute name="CardinalityOut" type="xsd:unsignedByte" use="required" />
    </xsd:complexType>
  - <xsd:complexType name="Component">
    - <xsd:sequence>
        <xsd:element name="Interfaces" type="gidecomposition:Interface"
          maxOccurs="unbounded" />
        <xsd:element ref="gidecomposition:Component" maxOccurs="unbounded" />
        <xsd:element name="Connections" type="gidecomposition:ConnectionType"
          maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="Name" type="xsd:string" use="required" />
      <xsd:attribute name="Id" type="xsd:unsignedShort" use="required" />
      <xsd:attribute name="LastModified" type="xsd:string" use="required" />
```


GIDE – An Insight in to composition …