CoreGRID
Institute on Programming Model

CoreGRID Industrial showcase
Open Grid Forum

Barcelona, Spain
4-5 June, 2008
Programming Model

❖ **GCM**: a CoreGRID programming model and methodology
  ❖ collecting partners experience
    ❖ UNIPI, INRIA, WWU, UNIPASSAU, VUA, QUB, UPC, HRLS, ULisboa, USannio
  ❖ component based, supporting autonomic computing
❖ STREP spin-off project: GridCOMP
  ❖ GCM reference implementation demonstrate the feasibility and sustainability of the approach
GCM
(coreGrid Component)
GCM genesis and goals

- Designed within CoreGRID NoE (6th FP)
- Mainly within the Programming Model institute
- Currently being developed within GridCOMP STREP (6th FP)
- Aimed at providing suitable tools for the efficient development of component based GRID applications.
GCM features

- Hierarchical components
- Collective communications and component interaction patterns
- Autonomic management of notable parallel composite components
- Advanced programming models
- Fractal based
Hierarchical components

Structural relationships

Functional network
Collective interaction

Client assembly

W1

W2

W3

Serv
Adaptivity
Autonomic management
Autonomic Computing

Managed components

- **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

Manager

- **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**: implement the selected strategy

QoS data

broken contract

next configuration
Management Orchestration

Cx = Component x
Cx', Cx'' = Instances of Cx
M(Cx) = Manager of Cx

Qos contract (from users)
Structural relationships
Management overlay network
Functional network
Behavioural Skeletons

❖ Parametric assemblies of components
❖ higher-order
❖ equipped with a pre-defined adaptation API & management strategy
❖ Behavioural skeletons abstract component self-management in component-based design as design patterns abstract class design in classic OO development
Farm (e.g. Dicom)

- change // degree
- new contract (e.g. Ts<k)
- get_service_time
- raise "contract violation"

File input → dicom → farm → screen

unicast → from_any
Dicom Example

- Medical images analysis
  - perform several kind of image segmentation to highlight suspect spots in medical images
  - sequential code developed by Pisa university clinic
- Parallelised with GCM by just plugging the sequential code into a Behavioural Skeleton
Progress

component model methodology programming tools NF & F features

monitoring API reconfiguration API passive BeSke

management policies QoS contracts manager engine

management co-ordination mechanisms and policies many open problems

coremodel area of interest

naming communication deployment sharing

middleware

CoreGRID kick-off 04

GridCOMP kick-off 06

adaptive components now

autonomic components

autonomic applications
More demos on demand
please ask us
Component, services or both?

- We re-defined and implemented autonomic BeSke in SCA/Tuscany
  - proof-of-concept implementation
  - JBoss rule-based manager
- Few differences
  - manager: JBoss rules vs POJO code
  - protocols: standard XML/SOAP vs Proactive
  - binding: static vs dynamic
- Proposal for standard extension
  - dynamic binding of components
  - Tuscany people shown interest