Abstractions and Models for Developing Distributed and Grid Computing Systems

José C. Cunha (jcc@di.fct.unl.pt)
CITI – Centre for Informatics and Information Technologies
Dep. Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Contents

Part I- Global Perspective
- Distributed, parallel and Grid computing
  - Distributed and Parallel Computing: common issues
  - Distributed and Parallel Programming Abstractions
- Application characteristics
- Grid abstractions and development issues
- Software Engineering issues and challenges
- Conclusions

Part II – Abstractions and Models for Distributed and Grid Computing
- Research at CITI at Uni. Nova de Lisboa
- Design Patterns and Operators for Grid Environments
- Group-oriented abstractions
1- Abstractions and Models

To capture common properties to a number of systems...
To focus on the fundamental aspects and ignore / hide the details...
To enable reasoning on fundamental properties...
Layers of Abstraction

- Application-level:
  - For each Application Domain / Classes of Applications
  - Problem-Solving Environments
- High-level Programming Abstractions and Models
- Intermediate Frameworks and Programming Interfaces
  - To hide lower-level concerns
  - To promote incremental development
- Service and Resource Abstractions
Transparency vs Awareness

- Transparency and Awareness
- How to make the right choices at each layer of abstraction?
- Transparency has been revised as time passes:
  - Raw hardware, Digital logic, Instruction-Set, Assembly, High-Level Languages, ..., Operating Systems, ..., Text editors, processing tools, ..., Database and Transactions, ..., Virtual Machines and Networks, ..., Web and Grid computing ..., Problem-solving environments ..., Autonomic computing ..., Semantic Web and Grid ...
Distributed Computing

Parallel Computing

Grid Computing

Applications and Problem-Solving Environments
Common Issues

- Adapting to geographical distribution and mobility
- Exploiting functional decomposition
- Providing quality of service
- Aiming at high-performance

- Large-scale applications?
- Novel forms of collaboration?
- New problem-solving approaches?
Physical Distribution

-- architectures with distinct degrees of control and data distribution

-- distinct internal organisations in terms of processors, memories and interconnection networks

-- with varying characteristics concerning performance, scalability, and dynamic (eg failure) behaviours
Logical Distribution

Logically distributed entities:

processes, objects, agents, components,…, services,
interact and cooperate.

They are spatially distributed: each has a local name / address space and they interact through communication devices (memories, or communication channels, or remote entry invocations)

This is supported by multiple system layers:

- Application/algorithm
- Programming models and languages
- Operating system/runtime support and communication protocols
Parallel /Distributed Computing

I - Using multiple processors:
-- composition / decomposition
-- mapping onto distinct computing nodes

II - Coordinating cooperating entities:
-- cooperation with coordination
-- communication and synchronisation

III - Handling dynamism:
-- react (detect / recover) to ‘’unexpected’’ events
-- manage dynamic reconfigurations (failure, mobility), system- or user-driven
III - Handling dynamism (1)

- Traditionally: fault-tolerance, task spawning, load-balancing

- Fault-tolerance: some approaches:
  
  a) full transparency (or opaqueness!):
  the runtime support system does it:
     -- high cost / sometimes wasteful
     -- can alleviated through control over checkpointing

  b) full responsibility to the programmer:
  the runtime system detects/triggers exceptions
  the program provides the handlers

  c) a different approach: TRANSACTIONS
  atomic execution units wrt to concurrency and failure
  and distribution
  and parallelism
Handling dynamism (2)

- Manage dynamic reconfigurations:
  -- detect unexpected events in general
  -- solutions depend on the system structure and scale: hierarchies, domains, groups, etc.

- Dynamic modifications:
  --- in the structure:
    - entities enter and leave the system
    - topologies change
  --- in the behaviour
    - interaction patterns change:
      - synchronous vs asynchronous
      - direct vs indirect communication
      - push versus pull
Distributed computing
Distributed Computing

- Physically distributed computations and data

- Goals:
  - Adapt to geographical distribution (LAN or WAN)
    - Users / Access / Processing / Archiving Sites
  - Provide availability and reliability
    - Fault tolerance / Redundancy
  - Support Cooperation and Collaboration
  - Handle Dynamic Behaviour
    - Reconfiguration and Mobility
  - Provide appropriate levels of transparency
Dimensions

- Data and control distribution (partition and replication)
  
  Criteria:
  
  - Desired functionality
  - Performance and quality of service
  - Dynamic reconfiguration (fault-tolerance, mobility)

- Distributed computing models:
  
  Time: synchronous vs asynchronous

  Communication: shared vs distributed spaces
  
  one-one, multicast, message orderings, shared memory

  Failure models: communication, computation
Transparency or Opaqueness?
Fit to the application and technology constraints

- e.g. Failure handling
- e.g. Communication models: message, remote method invocation, physically shared memory, distributed shared memory
- Coordination in the presence of uncertainty: events, causality, consistent distributed views
- Interactions:
  - Loose vs tight forms of interactions / collaboration
- Databases:
  - Transactions: Atomic wrt Concurrency and Failure
  - Consistency of replicated data: one-copy serialisation
  - Mobility, disconnected operation: local views/ snapshots
Distributed computing

- Composition, to integrate distinct services, and components
- Decomposition, to distribute the computation and data
- Coordination of distributed processes
Coordinating cooperating entities

Coordination is based on a reconstruction / observation of the system state that is required for processes to take consistent decisions:

it requires ensuring appropriate event orderings:

a) total orderings may be required:
   to ensure fairness
   to support consistency in the management of replicated data
   to support consistent views of the system/application history

b) partial orderings:
   are ‘more natural’ as they reflect the inherent concurrency
   allowing parallelism and autonomy
   need to preserve causal precedence
Approaches

- **Coordination:**
  -- centralised versus decentralised

- **Based on communication models:**
  a) **Shared Spaces:**
     --- shared-memory (physical or virtual)
  b) **Distributed Spaces:**
     --- messages (1-1, 1-N, N-1, N-N)
     --- remote entry invocations (RPC, RMI)
(0) Events

- This is independent of having shared or distributed spaces

- We need event models, in order to manage dynamic behaviour
Overview of models

Shared spaces:

physical or virtual/logical

versus

Distributed spaces:

Remote procedure call (RPC or RMI)

Messages: in pipes, streams, channels

Direct: point to point

Indirect:

via mailbox/message queue

via dissemination:

broadcast

Multicast → group interaction
a) distributed processes
remote invocation.

c) distributed processes
message exchange
indirect

b) message
d) message exchange
broadcast

distributed processes
msg2

distributed processes
msg1
A – Approach

- Abstract global variables represent the state of the system and their values are assumed to be kept coherent (under user-agreed coherency model)...

- Underlying layers support the concept: through centralised or decentralised solutions (e.g., data replication)

- Examples:
  - Using virtual global queues in distributed algorithms
  - One-copy serialisation in distributed databases
  - Distributed shared memory
  - Shared tuple spaces
Sharing global structures
B – Approach

- Distributed variables are explicitly managed by distributed processes in distinct address spaces/locations/sites.
- Distributed processes run almost autonomously based on local decisions and partial knowledge, and coordinate joint decisions through explicit communication and synchronisation.
Explicit decentralised variables
Difficulties are well known:

- Asynchrony (communication delays, unbounded)
- Concurrency
- Unexpected message arrival orderings

**Uncertainty (in space and time) and partial knowledge of the system state:**

- Information dissemination may be late, obsolete or inconsistent wrt to event occurrence
- Multiplicity of decision(observation)/control locations ---&gt; non-determinism

- Highly dynamic behavior:
  - Processes enter and leave
  - Failures occur leading to the need of reconfiguring the system
Abstractions for Distributed Programming

1- Abstracting the physical infrastructure → a system model with the main elements, their intrinsic properties, and interactions characteristics.

   Abstracting Processes
   Abstracting Communication
   Abstracting Time

Build models as a combination of abstractions.

2- Build abstractions that capture recurring interaction patterns in distributed programs.

   Examples:
   - Broadcast: best-effort, reliable, uniform, causal
   - Shared memory: data consistency models
   - Consensus: total order broadcast, atomic commit, group membership, view synchronous communication

Parallel computing
Parallel Computing

- Goal: to reduce execution time, compared to sequential execution.
- Relies on multiple processors cooperating for the coordinated and simultaneous resolution of parts of a given problem.

Different kinds of Computer Systems/Architectures:
- Shared / Distributed memory multiprocessors
- Local computer networks and Clusters of PCs
- Large-scale distributed Grids

Parallel Programming requires abstractions for:
- Decomposing application in parts – tasks, processes
- Scheduling the tasks in parallel processors
- Coordinating the cooperation between tasks

SIWN
Jose Cunha
The potential for parallelism is `everywhere`: (in the Nature...)

- **Science and Engineering**
  - Fluid dynamics
  - Particle systems
  - Weather forecast and Climate modelling
    - Complex models
      - simulations with large amounts of data
      - autonomous processes to achieve desired global collective behavior

- **Economy and Finance:**
  - Financial models

- **Simulation of VLSI circuits:**
  - Test generation, fault diagnosis

- **Databases:**
  - Parallel search
  - Parallelism across transactions (multiple simultaneous users)
  - Search of solution / state spaces
  - Pattern recognition / image processing
  - Natural language processing (parallel text mining)

-etc.
Developing Parallel Applications is difficult

1. Costs of task decomposition and cooperation depend critically on the system layers:
   - Application, Algorithm, Programming Language, Operating System, Computer Architecture

2. Computing platforms change quickly:
   - Supercomputers, Shared / Distributed memory multiprocessors, LANs and Clusters of PCs, WAN, the Grid?
   - With distinct degrees of control and data distribution, distinct internal organisations (processors, memories and interconnection networks), and varying characteristics concerning performance, scalability, and failure behaviors

3. Few standard / robust software tools & environments

4. The developer must have deep knowledge on the application domain

Jose Cunha
Parallel Virtual Machine

Task → Task → Task

dynamic pool of virtual processors

Hardware/OS Platform
(parallel machine/LAN/cluster)
Diversity of approaches

- **Explicit parallelism**
  - Architecture level
  - Operating system level
  - Generic Library / API (Threads, PVM, MPI, OpenMP,...)
  - Higher level programming languages
  - Visual programming / Workflows

- **Implicit / Hidden parallelism**
  - Declarative languages: logic and functional
  - Specialised libraries (eg ScaLAPACK for Linear Algebra)

- **Problem-solving environments**
Abstractions for Parallel Programming

See:

*Patterns for Parallel Programming*,
*T. Mattson, B. Sanders, B. Massingill, Addison-Wesley, 2005.*

A Pattern-based methodology: four design spaces, to visit in a systematic and iterative way:

- **Find the Concurrency**
  - problem structuring to expose the concurrency
- **Algorithmic Structuring**
  - to explore the identified concurrency
- **Support Structures**
  - structuring the program and data structures
- **Implementation Mechanisms**
  - correspondence to programming models/environments
Find the Concurrency

Patterns for
1. Decomposition
   - of tasks
   - of data
2. Dependency analysis
   - task aggregation
   - task ordering
   - data sharing
3. Evaluation (according to well-identified criteria)

From: Patterns for Parallel Programming, T.Mattson, B. Sanders, B. Massingill, Addison-Wesley, 2005.
Algorithm Structuring

Refining the design to define a parallel program.

**Main algorithmic patterns:**

- **task-centric**
  - task parallelism
  - divide and conquer

- **data decomposition-centric**
  - geometric decomposition
  - data recursion

- **data / control flow -centric**
  - regular: pipeline
  - irregular unpredictable and dynamic flow: event-based coordination

*From: Patterns for Parallel Programming, T. Mattson, B. Sanders, B. Massingill, Addison-Wesley, 2005.*
Support Structures

Program structures
  SPMD
  Master/ Slave
  Loop Parallelism
  Fork / Join

Data structures:
  Shared data
  Shared Queue
  Distributed Array

Incomplete list...

From: Patterns for Parallel Programming,
T. Mattson, B. Sanders, B. Massingill, Addison-Wesley, 2005.
Implementation Mechanisms

Mapping to the available mechanisms in the programming language environments:

- Management of execution entities
  create, destroy, manage processes and threads
- Synchronization
  data consistency and coordination of the interactions
- Communication
  exchange information
- Dynamic management of events
  handle unpredicted events

From: Patterns for Parallel Programming,
T. Mattson, B. Sanders, B. Massingill, Addison-Wesley, 2005.
Composition and/or Decomposition?
Steps in the Design of Parallel Programs (*due to Ian Foster*)

Involves to partition or decompose but also to compose

- **Partitioning**
  - Decomposition of computational activities and the data into small tasks – paradigms – e.g. master worker, pipeline, divide and conquer, SPMD, and speculation.

- **Communication**
  - Flow of information and coordination among tasks

- **Agglomeration**
  - Tasks and communication structure are evaluated for performance and implementation cost. Tasks may be grouped into larger tasks to improve communication

- **Mapping / Scheduling**
  - Assigning tasks to processors such that job completion time is minimized and resource utilization is maximized.
Unifying concurrency, distribution, and parallelism

All dimensions centered around an abstraction for a computational entity:
process / agent / object / component / service
Approaches towards distribution

- Adapt previous centralised solutions
- Propose new models based on distributed coordination

Examples:
Managing passive entities (memory and data) is simple in a centralised single-user system.
It becomes more difficult with concurrent users.
Even more, if the system is decentralised, and managed by server agents with multiple data copies

→ More complex coordination is required to deal with data coherency
Approaches for work organisation and distribution

- To distribute specialised tasks to distinct entities, with a hierarchical coordination, with a reduced number of entities at each level; and some independence between entities: loose coupling.

- To divide a complex task into simple and elementary subtasks, and to distribute them through a varying number of workers, with coordination required, depending on the application;

- Critically depending on the application, number of entities, degree of interaction and task granularity, and nature of the underlying distributed architecture.
Examples of diversity of work organisations

**Pipeline/workflow**: work organised as chains of worker units, their effectiveness depending on continuous feeding of units

**SIMD/SPMD** – Single Instruction/Program Multiple Data: each worker does the same task on a different data object

**MIMD/MPMD** – Multiple Instruction/Program Multiple Data: different workers, different tasks

**Large-scale**: tens, hundreds, thousands,… of entities operating according to several organisations, dynamically configurable…

→ Open issues…
Application characteristics
Evolution of Application Characteristics

- Complex models – simulations
- Large volumes of input / generated data
- Difficult interpretation, classification and search
- High degree of User interaction:
  - Offline / online data processing / visualisation
  - Distinct user interfaces
  - Computational steering
- Multidisciplinary:
  - Heterogeneous models / components
  - Interactions among multiple users / collaboration
- Require parallel and distributed processing
Heterogeneous Components

- Sequential, Parallel, Distributed Problem Solvers (simulators, mathematical packages, etc.)
- Tools for data / result processing, interpretation and visualisation
- Online access to scientific data sets and databases
- Interactive (online) computational steering
Problem-solving perspective

Parallel Computing requires:

- Decomposing the application into parts
- Launching tasks in parallel processes
- Planning the cooperation between tasks

In general, this is very difficult... and requires expertise both in Computer Science and in the Application Domain...

→ Integrated environments for solving classes of related problems in each application domain

E. Gallopoulos, E. Houstis, J. Rice

Requirements:

- To meet more complex applications
- To ease the cycle of application development, deployment and control
- To integrate heterogeneous components into an environment
- To allow transparent access to parallel and distributed resources
- To support collaborative modeling and simulation
Problem-Solving Environments (PSE)

A different approach:

Integrated environments for solving a class of related problems in an application domain:

- Easy-to-use by the end-user
- Based on state-of-the-art algorithms, which are encapsulated in components (libraries, packages, class OO repositories)
- Also including development and runtime support tools

Application components and computational tools are integrated into a single unified environment

Jose Cunha
Complex cycle of activities for the user/application developer

1. Problem specification
2. Configuration of the environment:
   - Component selection (simulation, control, visualisation) and configuration
3. Component activation and mapping
4. Initial set up of simulation parameters
5. Start of execution, possibly with monitoring, visualisation and steering
6. Analysis of intermediate / final results
A PSE gives access to:

- Abstractions for problem specification and for problem solving (relevant to the application domain)
- Resource management
- Execution support services
Hierarchically composition
PSEs: Problem Specification / Solution

- Use
  - Visual environment / workflow to link software components
  - High-level language specification

- Recommender systems: to help user choose best way to solve problem and locate appropriate software
Grid Computing

- Enables more complex applications
- Real application needs
- Grid application profiles
- Multiple Grid views and perspectives
Applications challenges demanding more ambitious goals

- Enable ‘heavy’ applications in science and engineering
  - Complex simulations with visualisation and steering
  - Access and analysis of large remote datasets
  - Access to remote data sources and special instruments (satellite data, particle accelerators)
- Distributed in wide-area networks, and
- Accessed through collaborative and multidisciplinary PSE, via Web Portals.
Major Steps towards the Grid

1. Networking: TCP/IP
2. Communications: Internet and e-mail
3. Structuring and integrating access to Information: the Web
4. Global Computing and Collaboration
5. Integration with Mobile, and Sensor Networks
6. Semantic / Knowledge
The Grid

User

collaboration

scientific instrument

User

supercomputer

local networks

cluster
Concept of a Grid

Gathers a diversity of resources, distributed at varying scales:
- supercomputers and parallel machines, and clusters
- massive storage systems
- databases and data sources
- special devices

Provides globally unified access to virtual resources:
- Transient: to support experiments (computation, data, scientific instruments)
- Persistent (databases, catalogues, archives)
- Collaboration spaces
Grids and Conventional Distributed Systems?

What are the differences?

The distinctive aspects:

- Higher levels of the transparency for the end-user
- Higher levels of integration of services
- Virtualisation of resources

User Abstraction
Resource Abstraction
User

Access Control

Global Shell

A pool of virtual resources

- application services
- computation services
- dataset, data repositories
- information services
- storage services
- physical resources and devices
Instead of:

- Manually subdivide algorithms
- Manage their execution on separate machines
- Need to have a separate user login account in each machine
- Manually merge and integrate the data and results

Exploit Grid abstractions to the same, more or less automatically, in a virtualised environment:

- Single login access point
- Access to logical resources
Virtualisation:

– Generic approach to:
  - Allow logical access to types of remote, heterogeneous, and distributed resources
  - As if they were a single larger homogeneous resource, locally available
  - Applies to: computation, storage, and network resources … and to any other LOGICAL RESOURCE

– Dynamically adjust resource mappings to match application demands
Question:

Real applications’ needs

- Solve new or larger problems by aggregating available resources at large-scale
  - for bigger, longer experiments, and more accurate models
- Easier access to remote resources
  - a large diversity of computation, data and information services
- Increased levels of interaction for increased productivity and capability to analyse and react
  - enable coordinated resource sharing and collaboration across virtual organisations
Applications / User Profiles

- **Computational Grids:**
  - Provide a single point of access to a high-performance computing service

- **Scientific Data Grids:**
  - Access large datasets with optimised data transfers and interactions for data processing

- **Virtual Organisations and Collaboration Spaces:**
  - Access to virtual environments for resource sharing, user interaction and collaboration

- **Access large geographically distributed data repositories, e.g. for data mining --- Information and Knowledge Grids**

- **Interaction and Sensor Grids**
  - Remote and large-scale data collection
  - Real-time interactions for decision support
The Grid is an evolving field

- Multiple views, perspectives
- Concepts, models and architectures still being defined and tested
- Applications still emerging
- Wide variety of interests
- Focus on geographically distributed multi-organisation, utility-based, service-oriented computation, open environment, dynamic and unsecure
Diversity of Grid Views

- Services providing virtual access to:
  - Applications
  - Storage
  - Computation and data
  - Interaction and collaboration

- Examples:
  - Virtual access to distributed supercomputing
  - Explore multidimensional parameter spaces
  - Virtual access to scientific instruments
  - Scientific datagrids and large-scale virtual data stores
  - Resource sharing and collaboration in virtual organisations
Grid development issues
Grids are very complex systems

- Aim at providing unifying abstractions to the end-user and virtualisation of resources
- Manage a large-scale universe of distributed, heterogeneous, and dynamic resources
- Critical aspects:
  - Distributed
  - Large-scale
  - Multiple administrative domains
  - Security and access control
  - Heterogeneity
  - Dynamic
Distinct Grid Scales

- **Space scale**: Local, metropolitan, regional, national, global
- **Time scale**: logically aggregate resources for long or short periods of time
- **Crossing borders**: Resources can span a single or multiple organisations, or service provider spaces
Grid Architecture in some detail

- User interfaces and Grid portals
- Applications and PSEs
- Models, tools and environments for application composition, programming and deployment
- Grid operating environment (middleware):
  - Services and resource management, discovery and scheduling
  - Information registration and querying
  - Authentication, Security
  - Computation, data management, and communication
  - Monitoring, Quality of Service
- Heterogeneous resources and infrastructure
Implementing Grids… from bottom to top…

- Infrastructure
- Middleware
- Services
- Programming models, tools and environments
- Applications
Grid development

- Infrastructure and services for aggregating distributed resources for solving large-scale problems or enable large-scale remote collaboration

  - Requires standard open generic protocols and interfaces
  - Well-defined middleware protocols to control connectivity, coordination, resource management, security, and accounting
Dimensions

Need to provide:
- Abstractions and models for distributed application design and development
- Associated tools and environments
- Relying on support infrastructures

Developments in distributed - grid computing
- Towards more robust, stable and standard infrastructures and support architectures
- Setting up the basis to build higher level abstractions, models and environments
Preparing for application development

Currently, basic services layers are not completely stable, standardised or fully deployed, so, how do to develop and experiment with higher-level abstractions for application development?
How to Go From Higher-Level to Low-Level Abstractions

- **Higher-Level Application Abstractions:**
  - For each Class of Applications
  - For each Application Domain

- **Problem-Solving Environments Layer**

- **Intermediate Frameworks**
  - To provide stable, generic, extensible APIs
  - To hide lower-level concerns
  - To promote incremental development

- **Basic Services Abstractions**
Software Engineering issues and challenges

- Components and their Interactions
- Dynamic resource management
- SE lifecycle and abstractions
More Complex
Applications and Environments

- Large number of components
- Complex interactions
- Heterogeneity
- Dynamic configuration
1 - Software Architectures

- Specification of components, their composition and interactions
- Modeling and reasoning on global structure and behavior
- Specification languages:
  - for structure and behavior
  - incremental refinement and dynamic composition
2 - Coordination models

- Represent and manage interaction patterns among components
- Communication and cooperation models
- Consistency guarantees
- Abstract, logical, dynamic organisation models
  - Dynamic application structure, interaction patterns and operation modes
3 - Resource management

- Configuration of parallel and distributed virtual machines
- Resource discovery, scheduling, and reservation
- Execution and monitoring at local and large scales
- Quality of service
Dynamic characteristics

- Looking at the past:
  - Fault tolerance, Load balancing, Task spawning

- At present and in the future:
  - Changes in the configuration and availability of resources, variations of characteristics and behaviour
  - Changes at the application level: user control of a dynamic experiment / unpredictable appl. evolution
  - Flexibility to build PSEs
  - Mobility of agents and devices
Enable more ambitious application requirements

- Distinct operation modes (offline/online)
- Distinct user interfaces
- User / Agent driven control
- Dynamic modification of operation modes and interactions
- Multiple users concurrently join ongoing “experiments” with distinct roles (observers, controllers)
Problem-solving strategies with adaptive behaviour

Awareness to Quality of Service factors
- Management at intermediate layers
- By intermediate agents – planners
- Contract negotiation – Service Level Agreement
- Dynamic revision of plans
- Reconfiguration

Specify, compose, develop, understand dynamic distributed large-scale applications: models, languages, and tools

Autonomic computing at distinct levels of abstraction
Conclusions on Challenging Requirements

- Higher degrees of user interaction, increased flexibility in observation, control, or modification of application components.
- Multidisciplinary applications, interactions between distinct sub-models, and distributed user collaboration.
- Dynamic applications and environments, as new application components or system resources are dynamically generated, made unavailable, or mobile.
- Spatial distribution of application components and system resources, at small, medium or large scales.
Critical issue

- The uptake of Grid computing technologies will be restricted by the availability of suitable abstractions, methodologies, and tools.

- We need to identify:
  - software engineering techniques for large-scale distributed and Grid environments,
  - along with specialist tools,

- Specialist software is necessary:
  - to enable the deployment of applications
  - to facilitate software developers in constructing software components for such infrastructure.
Diversity of Models

- Shared Memory
  - Threads/OpenMP
  - Distributed Shared Memory
  - Global Spaces/Linda

- Message Passing
  - Parallel Virtual Machine / Message Passing Interface

- Combined Shared and Distributed Memory
  - OpenMP and MPI
  - Groups with distributed and shared-memory

- Transaction-based Programming

- Event Models

- Objects, Components and Services
  - OO and Component models
  - Services: Web/Grid Services

- Organisation and orchestration
**Metacomputing**

- Use of multiple platforms (or *nodes*) to seamlessly construct virtual environments.
- Transparent access to remote nodes with single login access point and authentication.
- Heterogeneous components: specific functionalities, distinct computational models (sequential, parallel, event-based, etc).
- Components running on a single node may use distinct programming models or combination of them.
- Interaction between nodes is mediated by middleware and services layer with distinct technologies.
- Service-oriented architecture based on known APIs.
Grid Programming Abstractions

Grid programs are distributed programs: they are composed of logical individual *nodes* or components which interact in a coordinated way:

- **nodes** - encapsulate computation or data access components performing individual parts of a whole application
- **integration** and **global coordination** of the individual nodes and their interactions, as a distributed computation

*(In the mid-90s, *Metacomputing* was a keyword)*
Grid Programming Abstractions?

Abstractions for:

\( node = \text{software components or services} \)

- node composition and structure
- coordination: concurrency, parallelism, distribution; communication and interaction
- node aggregation and hierarchies
- meta-level information
- node code instantiation
- node registration, search and discovery
- node execution management and control
- specification of QoS
Software Engineering Challenges

Suitable levels of flexibility in all stages of the software lifecycle:

- Application specification and design
- Program transformation and refinement
- Simulation
- Code generation
- Configuration and deployment
- Coordination and control of the execution

See:
Issues - 1

- Clear separation and representation of concepts:
  - Computation and interaction
  - Structure and behaviour

- Specification of multiple components:
  - Enabling alternative mappings
  - Varying degrees of automated processing
  - Supported by pattern and template repositories with relevant attributes
Issues - 2

- Mapping the programming models into the underlying computing platforms:
  - Interacting with resource descriptions and discovery services
  - For flexible configuration and deployment

- Coordination of distributed execution:
  - Allowing workflow descriptions
  - With adaptability and dynamic reconfiguration
Part II - CITI-FCT/UNL
CITI - Grid-related Projects

- **TransGrid**: Parallel and Distributed Computing with Computational Clusters and Grids, 2003-, CITI-FCT/UNL
  - Applications in collaboration
    - Collaborative mobile multimedia
    - Parallel text mining
    - Materials Science
    - Geological Science
  - Abstractions and Models
  - Tools and Environments

- **Software Environments for Grid Computing**
  - 2002-..., Bilateral Cooperation with U. Cardiff

- **Linux Cluster and Grid Computing**
  - 2004-. IBM Equinox SUR Programme (Education & Research)

- **High-Performance and Grid Computing (Joint Curriculum Development)** 2005-2008, EU Asia-Link Programme
TransGrid: a CITI Research Project

- To improve parallel and distributed environments for complex problem solving, in computational clusters/grids.

- Dimensions:
  - Applications
  - Abstractions and Models
  - Tools and Environments
  - Distributed Execution
1 - Application Classes

- Parallel Text Mining;
- Collaborative Mobile Multimedia;
- Distributed Simulation, Visualisation and Steering (Geological and Material Sciences)
Application Characteristics

- large volumes of data (text or images):
  - efficient search, parallel processing and input/output
- dynamic, distributed, mobile application entities:
  - appropriate structuring, interaction, coordination
- integration of distributed heterogeneous components in a highly interactive environment:
  - supporting dynamic reconfiguration of components, and execution at a small or large scale
- organisation, management, coordination in a distributed agent system:
  - dynamic organisation and intelligent agents.
2 - Abstractions and Models

- **Design Patterns** → to abstract commonly occurring structures and behaviours in distributed dynamic environments

- **Dynamic Groups** → organisation and cooperation paradigm in distributed systems

- **Distributed Agents** → to support planning, intelligent decision support, and intermediate between the user and the PSEs and the system levels
Abstractions and Models

- Design Patterns
- Dynamic Groups
- Distributed Intelligent Agents

How the above can be combined to allow systems to be modeled as groups of agents, which may sometimes exhibit well-identified patterns of structure, interaction, and behavior.

How this approach can contribute to ease the tasks of specifying, composing, developing, understanding dynamic, distributed, large-scale applications.
Design Patterns and Operators for Distributed Grid Environments

(1) Cecília Gomes (Cecilia.Gomes@di.fct.unl.pt)
(2) Omer F. Rana (O.F.Rana@cs.cardiff.ac.uk)
(1) José C. Cunha (jcc@di.fct.unl.pt)

(1) Universidade Nova de Lisboa, Portugal
(2) Cardiff University, Wales, United Kingdom
Outline

- Problem-Solving Environments
- A Pattern-based Approach
- An Example
- Extension of the Triana tool
- Ongoing Work and Conclusions
Workflows for e-Science

Workflows considered fundamental in building the bridge between

Higher-Level Application Abstractions
and
Intermediate and low-level Frameworks

For a survey, see:
Ian Taylor, Ewa Deelman, Dennis Gannon, Matthew Shields (Editors)
Workflows

- to support adequate abstraction levels for scientists
- to ease the tasks of application development, and the management of the life-cycle of scientific experiments
- to allow flexibility in the mappings between the application abstractions and the lower layers of the underlying computing platforms
- Currently a diversity of research projects contributing to a new generation of more flexible and dynamic workflow-based development environments
Workflows seen at a level of meta-composition of applications, providing a macroscopic view

Useful to help automating common tasks and control their dependencies

Useful to abstract the main building blocks, their interconnections, and their interactions

Useful to register and consult, and to reuse results of previous computations

Useful for annotations, semantically related to the application
Abstractions for structure and behavior: example of ongoing work

1. To capture Workflow Structure and Behavior
   a. Composition time ("build time")
   b. Execution time

2. An Approach based on Patterns/Operators, an extension to Triana

Joint work U.N. Lisboa and Univ. Cardiff

Cecilia Gomes
Omer Rana
José Cunha

Separation allows multiple "build time" mechanisms to be mapped to different run time mechanisms
Goal

To contribute to application development in Problem Solving Environments (PSEs) in the Grid context:

- through the reuse and flexible composition of typical “idioms”;
- by providing some level of application control (each “idiom” is directly manipulable, including at execution time).
Grids are complex environments
  - heterogeneous; dynamic

Build applications (e.g. scientific) that take advantage of the disparate collection of resources and services provided by Grid infrastructures.

Typical “idioms” are frequently recurrent in Grid applications:
  - an “idiom” captures common knowledge and experience and describes how a similar set of experiments are to be set-up and managed
  - it is advantageous to identify those “idioms” and reuse them

C. Gomes, O. Rana, J. Cunha
An approach for application development based on:

- Design Patterns as first-class entities
- Operators for pattern manipulation
- Separation of concerns: structural and behavioural patterns and operators
- A methodology supporting pattern-based application development, reconfiguration and execution control (either through a GUI or scripts)
Design Patterns as First-class Entities

- Design Patterns
  - capture common usage “idioms”
  - provide a mechanism to capture common ways to use software within a given application domain

- Patterns as first-class entities during all phases of an application life-cycle:
  - development units
  - execution control units
  - reconfiguration units

- Structural and Behavioral patterns for flexible composition and manipulation
Target users

- Computational scientists and developers who understand the computational needs of their application domain
- Enable them to exercise common usage scenarios within a domain:
  -- particular components
  -- their coordination
- Patterns as first class entities both at design, execution, and reconfiguration times

- Approach: Pattern templates are manipulated through Pattern Operators

- Two categories for flexibility: structural and behavioural
  - Structural patterns; structural operators.
  - Behavioural patterns; behavioural operators.
Categories of Pattern Operators

- Structuring and Grouping Operators
  - establish/modify the connectivity between components in patterns (e.g. increasing/decreasing the number of elements)
  - build hierarchical patterns

- Inquiry Operators
  - verify structural/behavioral properties (e.g. if a pattern is hierarchical)
  - support pattern comparison on consistency/compatibility (e.g. before replacing a pattern with another)

- Ownership Operators
  - enable/delegate access rights to a pattern (e.g. to define which users may manipulate a pattern)

- Global Coordination Operators
  - establish behavioral dependencies between elements within a pattern (e.g. apply/replace a behavioral pattern)

- Execution Operators
  - provide execution control (e.g. suspend/resume execution)
1. Deploy Structural Patterns,
2. Refine them through Structural Operators,
3. Use Behavioural Patterns to define control/data flow/interactions,
4. Use Behavioural Operators to manage execution
Approach

Structural Operators
- Structural Pattern Templates
  - Refine/inquire/compose

Behavioural Operators
- Control execution/change dependencies

Behavioural Pattern Templates
- Applied to Structural Configuration

Components/Tools/Services
- Used for the instantiation of Template Configuration

Structural Configuration

BEHAVIOURAL OPERATORS
- Produced transformation
  - Application direct manipulation

Application Configuration

BEHAVIOURAL OPERATORS
- Ownership operations

Running Application
- Launch execution

SIWN
C. Gomes, O.Rana, J.Cunha
Structural Pattern Templates

- **Encode component connectivity.** Ex: Pipeline, Ring, Star, Facade, Adapter, Proxy.
Allow to build applications in a structured way:

-- select appropriate set of patterns, combine them according to operator semantics,

-- define new patterns and operators found useful and add them to the environment
Structural Operators

- Manipulate and compose structural patterns keeping their structural constraints

- Examples:
  - Increase, Decrease,
  - Embed, Extract,
  - Group,
  - Rename/Reshape, …
Increase Structural Operator

Pattern

Real Subject
Proxy

Result Pattern

Real Subject
Proxy
Proxy
Proxy
Proxy

Increase(Pipeline, 2)
Increase(Proxy, 2)

SIWN
C. Gomes, O.Rana, J.Cunha
Embed Structural Operator

A star embedded in the first component placeholder of a pipeline template, building hierarchies
Behavioural Pattern Templates

Capture recurring themes in component interactions
Behavioural Pattern Templates

- Capture temporal or (data/control) flow dependencies between components.

- Examples:
  - Client/Server,
  - Master/Slave,
  - Streaming,
  - Service Adapter,
  - Service Migration,
  - Service Aggregator/Decomposer, ...
Behavioural Operators

Are applied over structural templates combined with the behavioural patterns, after instantiating the templates with runnable components.

Act upon pattern instances for execution control and reconfiguration.
Behavioural Operators

Examples:
- Start, Terminate,
- Log,
- Stop, Resume,
- Restart, Limit,
- Repeat, …
Structural and behavioural operators can be combined into scripts that can later be reused in similar application scenarios.
3- An Example

- Problem Solver
- Monitoring Service
- Steering Interface
- Database System

Ring Pattern

Pipeline Pattern

Producer/Consumer

Streaming

SIWN
C. Gomes, O.Rana, J.Cunha
3- An Example

Problem Solver → Monitoring Service → Steering Interface

Input data → Output data → Selected data

Proxy Pattern

Real Subject (Steering Interface) → Proxy → Proxy
An Example

Problem Solver \rightarrow Monitoring Service \rightarrow Steering Interface

Input data

Output data

Selected data

Adapter Pattern

Adapter

Adaptee (Monitoring Service)

Service Adapter

SIWN
C. Gomes, O.Rana, J.Cunha
3- An Example

Star Pattern

Database (master)

Database (slave)

Database (slave)

Database (slave)

Problem Solver Interface

Steering Interface

Database System

Master/Slave
3- An Example

Problem Solver
  Database (master)

Adapter
  Adaptee (Monitoring Service)

Steering Interface
  Proxy
  Proxy

SIWN
C. Gomes, O. Rana, J. Cunha
Reconfiguration Capabilities (this is still ongoing work)

- Pattern-based reconfiguration
  - independent modification of structure and behaviour
  - first-class manipulation allows reconfiguration automation (pre-defined new dependencies)
- Reconfiguration at development time
- Run-time reconfiguration
4- Implementation

1. Extend an existing environment with support for patterns and operators
2. Enable management of applications by mapping Behavioural patterns / operators onto Resource Management Systems
Triana: http://www.trianacode.co.uk/

- The Triana tool is a component-based problem-solving environment for distributed environments (developed at Cardiff University).

- Includes a large library of pre-written analysis tools (units) and the ability for users to easily integrate their own tools.

- Already used by scientists for a range of tasks, such as signal, text and image processing.

- Generic environment for creating scientific workflow in multiple domains combining local and remote tools and services such as Web and Grid Services, and Peer to Peer computing.

- Provides support to create new units, adding them to a toolbox, a unit composition editor, and interfaces to RMS.
Applications are built by connecting services available in a toolbox. The execution follows the dataflow model.
GridLab Implementation

http://www.trianacode.org/

- GAP Interface
- JXTAServe
- P2PS
- WServe
- Gridlab Services
- OGSA + Services
Composition Environment
(Problem Solving/Portal Environment)

Components (Units)

Patterns

Operator Library

Distribution Interface

GAP/GAT (Triana/GridLab)

DRMAA

CoG Core/OGSA

Web Services

Peer-2-Peer (P2PS/JXTA)

Local Resource Managers
Extending TRIANA

- Patterns are treated as group units
- Each group element is a dummy unit to be instantiated with executables from the toolbox

- Extending Triana: current prototype
  - Structural patterns
  - Structural operators
The User’s View

- To configure and execute an application using the patterns library:
  1. Select a structural pattern (eg pipeline)
  2. Refine the chosen pattern with structural operators
  3. Select a behavioural pattern to specify interactions (eg dataflow)
The User’s View

4. All place-holders must be instantiated with units or group of units
5. Use a behavioural operator (eg start) for controlling the execution…
The System View

- Patterns are available at the GUI to be composed / manipulated through operators.
- Behavioural patterns map to a RMS that coordinates the execution.
- A ``pattern executor`` enforces the selected behavioural pattern at each element.
- Operator invocations are passed to the RMS interface API.
Patterns and Operators in Triana
The System View

- The GUI sends the user’s requests for execution to the local TRIANA Service, as a taskgraph.
- Execution may be local or distributed.
- Each TRIANA service may delegate execution to a local RMS.
To Conclude

- Mechanisms to structure application construction in PSEs
- Structure provided via “Patterns”
- Software Engineering support:
  - Structural and Behavioural Patterns
  - Structural and Behavioural Operators
- Ongoing work: implementation of behavioural patterns and operators.
Further Work

- Coordination and dynamic issues
- Experiment with selected application classes
Comparison

- Distinct from other approaches:
  -- structural constraints are defined separately from behavioural constraints
  -- our concern is to link patterns with specific resource managers and composition tools
Patterns/operators implemented in Java

Behavioural patterns/ operators in scripts are seen as pre-defined units in TRIANA: the user does not need to know their implementation in some language or scripting tool.

But a developer may …
Further Reading

- Pattern/Operator based Problem Solving Environments, Cecilia Gomes, Omer F. Rana, and Jose Cunha, EuroPar 2004, Springer Verlag, Pisa, Italy, August 2004.
Group Abstractions for managing Distribution, Dynamism and Scale
Groups of distributed entities

A group is a set of cooperating entities:

- With a unique and global name
- Addressed as a single entity in the system
- Members can enter and leave dynamically

Member share common and consistent views of the group’s history:

- Current membership
- Perceived events
  - Changes to the group state (entering/leaving)
  - Interactions (message delivery and other interactions)
Group applications

To support services in distributed systems:
- members cooperate to handle requests and to ensure:
  - fault-tolerance
  - parallelism
  - improved performance through replication
  - functional decomposition, internal to the group
- clients address the group as an atomic entity in a transparent way, through a well-defined interface
Groups as structuring units may include processes or other groups.
Dynamic Groups: not a new idea

Long-term research on Process Groups: School of Cornell University - Kenneth Birman (Virtual Synchrony models) and many others.

- But still many aspects largely unexplored:

- Organisation and cooperation paradigm to support

  - Scale, dynamism, and mobility, eg for local or ad-hoc dynamic aggregation of distributed / mobile entities

  - Interaction and coordination in small, medium, or large scale organisations.. Exploit forms of shared knowledge, and information, and trust relationships among group members, and for specialisation of services and cooperation (e.g. collective communication / shared memory / load balancing / fault tolerance)

  - Units of system or application composition to help build and manage complex and dynamic organisations / hierarchies

  - Collaboration:
    - Common computational or communication behaviors
    - Common goals in a society of agents
    - Need of sharing common resources and information
    - Cooperation towards providing common service functionalities with specific constraints (Performance, QoS, Cost parameters)
Groups for structure and organisation (1)

- Collections of agents which share common attributes
- Common logical characteristics shared by group members
  - Common computational or communication behaviors
  - Common goals in a society of agents
  - Need of sharing common resources and information
  - Cooperation towards providing common service functionalities with specific constraints
    - Performance
    - QoS
    - Cost parameters
Groups for scalability(2)

- By allowing hierarchies of entities where a group member can be an individual entity or another group
- Important in large-scale and complex organisations
- Allowing confinement of local and global policies
- And more flexible and efficient forms of communication and information dissemination
Groups for modelling dynamic systems(3)

- By providing consistency of views among the group members
- By supporting forms of cooperation among group members, including a shared group state
- Or to manage components with common properties
- By allowing dynamic change of group membership
Groups as units of system composition

- Groups can appear at distinct abstraction levels:
  - At application level
  - At programming level
  - At system level

- Groups can be considered as programming units and used to build hierarchies:
  - From its outside, a group can be viewed as an object, an agent, or a service, through an well-defined interface (like a set of methods, or ports), and with an internal behavior, hidden from the outside.
  - Separation between the group interface and its internal behavior allows implementing local policies, internal to a group, in a transparent way.
Groups as units of system composition

- A group can support a reactive or a pro-active, and goal-oriented behavior.
- It is possible to organise a distributed application or system in terms of collections of multiple groups, each responsible for a local service and policy, and globally managed by having global coordination and policies for overseeing and deciding on global strategies.
Research issues

- To investigate a group-based framework by providing a two-level approach:
  - [Group specification and management]: for the organisation in terms of groups of entities (as objects, agents, or services), where a group is a structuring unit:
    - With a public interface
    - And well-defined internal behavior
  - [Dynamic group discovery]: to dynamically discover and identify groups in a distributed environment, being guided by a definition of the common attributes which represent common characteristics of each group
- The two levels are orthogonal and can be developed independently
To exploit group concepts, in order to handle scalability, dynamism and mobility.

A high-level group-oriented model:
- for the dynamic organisation of distributed agents
- Integrating point-to-point, multicast, and logical shared-memory interaction models
distributed processes within a group

internal interaction via messages or via shared space

group interface
Group-oriented Abstractions and Models

Groups at distinct abstraction levels:
- As Application Units
- As Programming Units
- As System Units

GroupLog, an abstract model: agents, groups, forms of interaction
Distinct instances, at distinct abstraction levels:
  - Logic-based instance: GroupLog
  - Java-based instance: JGroupSpace with cluster implementation
  - MAGO: distributed interactive applications
Previous and ongoing work

- **GroupLog**: an abstract model, defines the basic elements: agents, groups, forms of interaction
  - Designed to allow distinct instances, at distinct abstraction levels
  - A logic-based instance of the model:
    - distributed Prolog based
    - Implemented on top of PVM-Prolog
  - A Java-based instance of the model:
    - Distributed implementation
    - Built on top of JavaGroups (a protocol composition framework)
  - With distributed, cluster-based implementations
  - Being extended for large-scale distributed systems

- Distributed problem solving with GroupLog:
  - Examples illustrate the applicability to a large diversity of distributed applications
  - Ongoing work exploits GroupLog for collaborative mobile applications

SIWN
Jose Cunha
A **GroupLog system**: a collection of distributed agents, able to:
* Communicate through interface predicates
* Access the **Group Shared State**
* **Join groups** to participate in coordination activities
Another perspective(6)

- Many distributed applications require the ability to capture and identify common attributes and their changes related to distributed and dynamic entities evolving in large-scale environments (like the Grid).
- The need to identify such attributes and their changes can become a critical concern, for example:
  - For intelligent strategies for resource management, depending on changing cost and resource usage.
  - To dynamically form ad-hoc groups:
    - As spontaneous identification of communities of interests (e.g., geoproximity between mobile users).
    - As dynamic definition of common interests, in reaching common goals, sharing common knowledge and functionalities, and contributing to common tasks.
Another perspective(7)

- The dynamic identification of groups as emerging from dynamically identified patterns of behavior or from the intention of pursuing common goals
- This can become a powerful mechanism to guide strategies for autonomic management of complex distributed systems and applications
Group-based approach for modelling interactive distributed applications

Main goals:
- capture the group dynamics
- manage the interactions between group members
- offer different kinds of communications mechanisms
- share data inside each group
- create explicit and implicit groups

Application:
- Interactive Narrative Storytelling
  - Project: InStory with IMG CITI Stream
MAGO - Modeling Applications with a Group-based Approach
MAGO - **Modeling Applications with a Group-based Approach**

- MAGO proposes a model to ease the development of interactive distributed applications.
- The model encompasses a set of primitives and services specialised for environments supporting group-based applications.
- It allows the organisation of systems in terms of multiple groups, each group being considered as a unit of system composition with a well-defined interface.
- Multiple entities can dynamically enter and leave groups, and distinct forms of communication among group members are allowed:
  - **direct**: direct communication between entities
  - **events**: multicast information to a group
  - **shared space**: share information inside each group
- Information system maintains the data associated to members and groups.

SIWN
C. Morgado, J. Cunha, N. Correia
MAGO Layered Architecture

- Applications

MAGO Model

- Entity management
- Group dynamics
- Communication interactions

Middleware support
(groups + events + shared space)

Information system access

Information system

Infrastructure
(network + operating system)

C. Morgado, J. Cunha, N. Correia