Abstractions for Organising Dynamic Distributed 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

1- General comments on abstractions for distributed computing
2- Organisation → Group-based abstractions

The focus is on open issues and challenges.
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...
To ease application / system development
correct / productive / efficient

- Transparency and Awareness
- How to make the right choices at each layer of abstraction?
- And how efficiently can we provide it to the user?
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
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**
Common Issues

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

- Large-scale applications? → specific abstractions?
- Novel forms of collaboration?
- New problem-solving approaches?
Need to abstract \(\rightarrow\) Logical Distribution

Logically distributed entities:

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

They are spatially distributed: each has a local name space and they interact through communication devices

Abstractions appear at multiple system layers:

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

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

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

III - Handling dynamism:
-- react (detect / recover) to unexpected events
-- manage dynamic reconfigurations, system- or user-driven
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
Distributed computing

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

- Coordination:
  -- from centralised to 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)
a) distributed processes
remote invocation

b) distributed processes
message exchange
direct

message msg1

c) distributed processes
message exchange
indirect

d) distributed processes
message exchange
broadcast

DPA Euro-Par08 Workshop
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 ---> 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, virtual synchrony

Parallel Virtual Machine

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**
Grids and Distributed Systems?

Can we speak of 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
A pool of virtual resources

application services
computation services
dataset, data repositories
information services
storage services
physical resources and devices
Need of abstractions \(\rightarrow\) Applications 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

- **Interaction and Sensor Grids**
  - Remote and large-scale data collection
  - Real-time interactions for decision support
Unifying concurrency, distribution, and parallelism

All dimensions centered around an abstraction for a computational entity:
process / agent / object / component / service
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
Composition and/or Decomposition?
Programming Abstractions

Distributed programs: composed of logical components which interact in a coordinated way:

--- encapsulate computation or data access components performing individual parts of a whole application (or a coupled application)
--- integration and global coordination of the individual nodes and their interactions, as a distributed computation
Distributed Programming Abstractions?

Abstractions for:

( = software components or services)
- composition and structure
- coordination: concurrency, parallelism, distribution; communication and interaction
- aggregation and hierarchies
- meta-level information
- code instantiation
- registration, search and discovery
- execution management and control
- specification of QoS, SLA
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;
- Distributed Simulation, Visualisation and Steering (Geological and Material Sciences);
- Collaborative Mobile Multimedia;
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 application levels and the system levels

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.
Abstractions for structure and behavior

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

3. 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

DPA Euro-Par08 Workshop
Group Abstractions for managing Distribution, Dynamism and Scale
Approaches towards distribution

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

A spectrum of solutions from centralised to decentralised solutions

→ 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, SIMD/SPMD, MIMD/MPMD

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

→ Open issues...
Several applications involve groups of HUNDREDS / THOUSANDS of intelligent software agents, and other entities (persons, robots)... that cooperate towards complex goals.

How they coordinate their joint action in a coherent and effective way, is a great challenge.

These systems are also dynamic and unpredictable, that is unexpected events occur that need reaction.

See:

Coordination of Large-Scale Multiagent Systems
P. Scerri, R. Vincent, R. Mailler (Editors)
Springer, 2006
Preliminary notes

Existing approaches work well for small scale groups (tens) and reached a significant level of maturity but more difficulty for coordination at large scale.

On the other hand, new approaches – promising large scale, such as swarm-based groups, still lack an adequate control, at a macroscopic level, on the guarantees of coherent collective emergent behavior, for generic cases. Only for very specific case studies, is such a relationship understood.
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 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 may address the group as an atomic entity in a transparent (opaque) way, through a well-defined interface
Dynamic Groups: not a new idea
Long-term research on Process Groups: School of Cornell University - Kenneth Birman (Virtual Synchrony models) and many other efforts (CSCW, Multiagents, OO, and P2P)

- 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 (collective communication / shared memory / load balancing / fault tolerance)
  - 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)
  - Units of system or application composition to help build and manage complex and dynamic organisations / hierarchies
Groups for scalability

- 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
  - Exploit varying scale
  - Exploit locality (physical and logical)
  - Exploit specific semantics contexts for interaction with distinct consistency guarantees
  - Exploit knowledge-based approaches
Groups as structuring units may include processes or other groups.
Example- Flexible coordination of large scale groups(1)

Example of work at Pittsburgh and CMU (P. Scerri et al)

See:

Coordination of Large-Scale Multiagent Systems
P. Scerri, R. Vincent, R. Mailler (Editors)
Springer, 2006
Example- Flexible coordination of large groups

Traditional approaches:
- each agent has an explicit detailed model of the others and of the joint activity of the group which is used to decide how to achieve joint goals

Not relying on accurate models: reduce communication
- algorithms that lead to cohesive and robust groupwork with high probability

- Example of work at Pittsburgh and CMU (P. Scerri et al)
Flexible coordination of large scale groups (3)

Example of work at Pittsburgh and CMU (P. Scerri et al)

Introduce an acquaintance network

logically connects the agents and is neutral to the subgroups organisation

it is a *small worlds network* (any two nodes are separated by a small number of neighbors)

Information is shared among agents who are neighbors

Exploit aspects of social networks to increase efficiency of coordinating large groups

See:

*Coordination of Large-Scale Multiagent Systems*

P. Scerri, R. Vincent, R. Mailler (Editors)

Springer, 2006
Flexible coordination of large scale groups (4)

Example of work at Pittsburgh and CMU (P. Scerri et al)

High-level group behavior – a logical model of group as an acquaintance network -- groups organised as social networks and subgroups detecting events in the environment, leading to plans to achieve the global goals of the group.

The group finds subgroups to work on those plans and within subgroups agents communicate to keep accurate models for cohesive group behavior.

Across groups agents communicate the goals of subgroup so that interaction among subgroups can be detected/conflicts resolved.
Another perspective

- 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.

- 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., geog. proximity between mobile users).
    - As dynamic definition of common interests, in reaching common goals, sharing common knowledge and functionalities, and contributing to common tasks.
Example – OpenMAS Approach (1)

Abramson and Mittu, US Naval Res. Lab.

Raises a number of issues due to the characteristics of large scale distributed open environments:

1- Group formation
2- Role allocation
3- Synchronisation of beliefs
4- Communication selectivity
5- Information sharing

See:

Coordination of Large-Scale Multiagent Systems
P. Scerri, R. Vincent, R. Mailler (Editors)
Springer, 2006
Research in GroupLog

José Cunha (Coordinator)
Fernanda Barbosa (GroupLog in Logic),
Carmen Morgado (MAGO),
Jorge Custódio (JGroupSpace)

- Groups, as an organisation and cooperation paradigm in distributed systems.
- A large complex system organised in groups, which may be further structured forming hierarchies.
- Interactions among group members are more easily managed due to its smaller scale, thus enabling more appropriate coordination paradigms
  - Groups encapsulating confined interaction spaces
  - Integrating point-point, one-many, and shared-memory interaction models
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, interaction space

Distinct instances of the model, at distinct abstraction levels:

- Logic-based instance: GroupLog
- Java-based instance: JGroupSpace
- MAGO: collaborative interactive applications
A **GroupLog system**: a collection of distributed entities, able to:
* Communicate through interface predicates
* Access the **Group Shared State**
* **Join groups** to participate in coordination activities
MAGO - Modeling Applications with a Group-based Approach
MAGO Layered Architecture

APPLICATIONS

MAGO MODEL
- Entity management
- Group dynamics
- Communication interactions

Middleware support
(groups + events + shared space)

Implicit groups
Information system access

Information System

Infrastructure
(network + operating system)
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
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**
Thank you.