J Grid Computing (2006) 4: 395–412
DOI 10.1007/s10723-006-9049-1

MAGDA: A Mobile Agent based Grid Architecture

Rocco Aversa · Beniamino Di Martino · Nicola Mazzocca ·
Salvatore Venticinque

Received: 18 October 2005 / Accepted: 13 May 2006 / Published online: 27 June 2006

Abstract Mobile agents mean both a technology and a programming paradigm. They allow for a flexible approach which can alleviate a number of issues present in distributed and Grid-based systems, by means of features such as migration, cloning, messaging and other provided mechanisms. In this paper we describe an architecture (MAGDA – Mobile Agent based Grid Architecture) we have designed and we are currently developing to support programming and execution of mobile agent based application upon Grid systems.

Key words mobile agents · Grid · heterogeneous and distributed systems

R. Aversa · B. Di Martino (✉) · S. Venticinque
Dipartimento di Ingegneria dell'Informazione,
Seconda Universita degli Studi di Napoli,
via Roma, 29, 81031 Aversa (CE), Italy
e-mail: beniamino.dimartino@unina.it

R. Aversa
e-mail: rocco.aversa@unina2.it

S. Venticinque
e-mail: salvatore.venticinque@unina2.it

N. Mazzocca
Dipartimento di Informatica e Systemistica,
Università Federico II di Napoli,
via Claudio, 21, Napoli (NA), 80125, Italy
e-mail: nicola.mazzocca@unina.it

1. Introduction

The increase in size and performance of computer networks is both cause and effect of their ubiquity and pervasiveness. This means that users are able to exploit network connectivity without expensive add-on and wherever they are, so they draw industries to install new networks ad to provide new services. Another relevant phenomenon is the increasing availability of handheld mobile devices and their increasing capability. In this context exploitation of distributed resources and services should be addressed. New programming paradigms are being designed and tested in order to develop models of applications that exploit the opportunities of these systems and overcome their drawbacks. One of the most important targets is to enable seamless utilization, interoperabilty and composition of the distributed resources and services. Technologies and standards for addressing this task, commonly named as *Web Services*, are promoted by W3C (World Wide Web Consortium) and are fast developing and achieving robustness and wide adoption. They define an interaction model to deploy business to business application in a Virtual Organization and provide a technology to achieve self-describing, discoverable services and interoperable protocols based on XML messages exchange. Grid community has recently decided to adopt Web Services technology in order to gain independence from the implementation of any service. The alignment – and augmentation – of Grid and Web services technologies is the aim of the *Open Grid Services Architecture* (OGSA) effort [3]. Anyway Web Services represent just a technology to build uniform interfaces which support a uniform access to the deployed services. Their adoption does not solve the still open issues arisen in Grid computing. A lack of functionality is still present in available Grid Environment [39, 41]. Missing services include resource brokering, automatic software dependency analysis and installation, configuring execution environments [38], and policy-based access control. An execution service to reliably execute complex jobs in a Grid environment even is not available. The most known example of Grid environment is Globus [2]. Even the Globus approach [40] focuses on low-level protocols, arguing that the creation of robust, core low-level protocols enables other projects to create higher-level tools and protocols that will provide advanced services. We propose in this paper an approach based on adoption of techniques for code and data mobility (commonly named as 'Mobile Agents technology'), in order to augment existing Grid frameworks with features of dynamic workload balancing, reconfigurability, reliability, support to distributed programming, monitoring and management of Grid resources etc. Code mobility can be defined informally as the capability to dynamically change the bindings between code fragments and the location where they are executed [4]. The mobile agents technology has gained interest in the last years in parallel and distributed programming because of its flexibility. This technology allows the user to exploit the remote resources of a distributed system, overcoming the limitation of its connection mode (device and communication channel), moving computation where data and any kind of resource are available. In this paper we discuss how mobile agents technology can improve existing Grid technology and can be integrated into existing Grid frameworks. We introduce a Mobile Agent based Grid Architecture (MAGDA), we have designed and we are implementing, conceived to support programming and execution of Grid-oriented agent based applications. The paper is structured as follows: Section 2 introduces related work about mobile agents for Grid; Section 3 describes pros and cons of the pros and cons of integration of Mobile Agents into Grid technology; Section 4 presents a detailed description of the framework we are developing; in Section 5 we provide an evaluation of MAGDA, by presenting a case study application its execution on a production Grid System; finally we come to conclusion.

2. Related and Complementing Work

Mobile Agents [5] are the last evolution of Mobile Code based systems. They add mobility to the set of features, such as reactivity, proactivity, communication and social ability, which characterize common software agents systems. A mobile agent is an *agent*[1] able to migrate across the network together with its own code and execution state. The idea of a self-controlled execution next to the data source has been proposed [6] as the new wave to replace the client–server paradigm with a better, more efficient and flexible mode of communication. The Mobile Agent paradigm involves the migration of a whole existing computational component together with its know-how and its intermediate results from a host to another where the necessary resources are available. As it is shown in Figure 1 the agents perform the computation were the resources are located thus optimizing the execution time and reducing the interactions from remote(we need just to send the

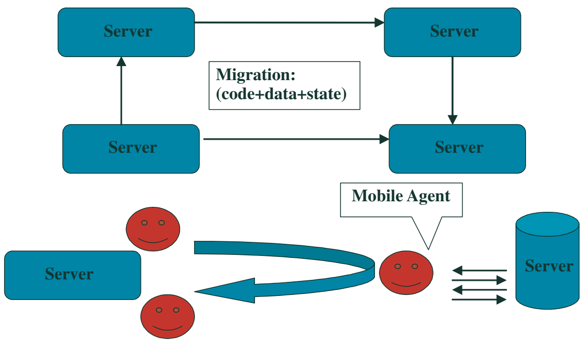

Figure 1 The mobile agent programming model.

[1] For a definition of the concept of *agent* see [44].

agent code and eventually to collect results). Mobile agents provide a mechanism for the delivery and execution of tasks on remote machines. Several mobile agent packages currently provide the basic functionalities together with other features about security, communication, user interface and management. The agent migration is more similar to process migration than to an applet download [4]. However the image of a process is strongly tied to the Operative Environment inside which it has been created, so it is not feasible to move such a process across different architectures. Two forms of mobility characterize the migration of agents a weak mobility and a strong mobility.

– `Strongmobility` is the ability of a Mobile Code System (called strong MCS) to allow migration of both the code and the full execution state of an agent.
– `Weakmobility` is the ability of a MCS (called weak MCS) to allow code transfer across different CEs together with the intermediate values of the some data which are relevant to resume the status of the computation. (the activation stack and complete image of the process could not be migrated, the application is resumed, not the process)

In this work we will deal with object migration that makes it possible to move objects among address spaces, implementing a finer grained mobility with respect to process-level migration even if this does not allow for strong mobility, but only for weak mobility. Agent technology, and mobile agent in particular, has recently gained interest in the Grid community [33, 36, 37]. Many works on design of agent based Grid platforms focus on the implementation of basic services such as routing and handling of FIPA ACL[2] messages [8], load balancing [9], fault masking [7] and service discovery [10]. An important component of resource sharing is resource description, discovery and interoperability. A Grid-based architecture for Multimedia services management is presented in [11]. Especially [10] defines an architecture of a services discovery functionality implemented through mobile agent technology. In [13] software agents are utilized in order to manage and assign resources and to distribute applications and data. A Java micro-kernel allows to hold and release the system resources for the specific application. Mobile agents take care to start applications and to monitor them. An agent is able to take a snapshot of the target application in order to recover it on a different host. A similar architecture is described in [14] where all the features of job migration are encapsulated in a user program wrapper that is implemented on a Java layer between a mobile agent and the corresponding user program. [15] and [16] also focus on monitoring and management of Grid resources. Some other works deal with parallel programming of distributed applications as in [17] where the agent based Messenger environment is used to implement two parallel programming skeletons, which provide the developer with some facilities for automatic distribution of tasks. In [18] a scheduling dynamically arrange and optimize the distributions of the working agents across the network according to the throughput of communication among them. A Problem Solving Environment built on an agent Grid is described in [34]. Many other issues dealing with Grid and mobile agents programming are addressed in literature such as security [19] and exploitation of Grid capability from heterogeneous and mobile devices [20–22].

[2] FIPA is the Foundation for Intelligent Physcal Agents (http://www.fipa.com). Its specifications define the standard to design and build agent platforms and applications. ACL is the Agent Communication Language. It has been defined in order to support the communication among agents developed by different parties.

3. Current Grid Technology Limitations and Opportunities for Mobile Agents

A mobile agents based middleware could be integrated within a Grid platform in order to provide each architecture with the facilities supported in the other one. Here we introduce some limitations of current Grid middleware and introduce relevant characteristics of Mobile Agent technology which can be exploited to develop Grid services and applications. A lot of open issues still need to be addressed in developing Grid services as the current available Grid environments lack relevant functionalities.

- In general current middleware are not able to migrate an application from one system to another as there are a lot of issues to be addressed [42]. Some examples are the differences across resources in installed software, file system structures, and default environment settings. On the other side reconfiguration is a main issue to implement dynamic load balancing strategies.
- Based on a message passing paradigm such as MPICH_G [1] Grid environments do not provide high level of abstraction compared to the current practices mostly based on traditional message-passing primitives.
- Current Grid middlewares still lack mature fault tolerant features [43]. Most of the earlier proposed facilities have been either designed for local area networks or to handle small number of nodes and hence lack in areas such as scalability, efficiency, running times etc.
- Current information and monitoring frameworks do not scale to Grid level or are focused on specific aspects. Monitoring and execution support with adequate checkpointing and migration support also still lack.[3]
- Currently, there is neither a coherent and generally accepted infrastructure to manage resources nor are there efficient coordination algorithms that suit the complex requirements of a large scale Grid environment with different resource types [38].
- Globus and other Grid platform do not provide a scheduler [39], but rather rely on the client operating system scheduler or batch schedulers such as OpenPBS[4] to handle local scheduling activities. Global scheduling between Grid processes can be provided by meta-schedulers, such as Condor-G.[5]

We aim to address the issues introduced above providing advanced services which are implemented through the ready mechanisms made available by the mobile agent technology such as cloning, migration, persistence, etc.

- Mobile agent technology is very flexible as it supports run-time mobility through both push and pull interaction models. We mean that it is possible either to download an agent to reconfigure the client and to upload the execution wherever there is an agent enabled host that provide computing power. It allows to migrate the execution according to dynamic changes in Grid status; suspend or resume the execution of time consuming applications; exploit new nodes by a run-time reconfiguration that provides, just when it needs, with those components which are needed to take part in a distributed computation;
- Different message delivery mechanisms blocking or non-blocking send and receive primitives allow to develop complex client–server or peer-to-peer applications. Client server or peer to peer paradigms can be implemented by different kind of communication facilities. Furthermore mobile agents dynamically optimize communications: Collaborative agents can be moved in a distributed environment in order to re-shape the initial communication pattern.
- Persistence, cloning and migration mechanisms which are provided by the mobile agents technology can be exploited to improve reliability by replication [7].
- Migration can be exploited to monitor remote resources. Agents can move to remote destinations to evaluate system parameters which allow to characterize the status of the target machines. Moving themselves to the front-end nodes they could be able to evaluate the real configuration of hidden resources.
- Moving code where data resides allows to reduce the traffic in the network and to remove the overhead due to the latency of communication. This allows also to dynamically optimize communications which could be affected by new traffic conditions or by unbalance of the distributed computation. Cooperation a

[3] See the objectives of the CoreGRID project http://www.coregrid.net.

[4] The OpenPBS Project: http://www.openpbs.org.

[5] The Condor Project: http://www.cs.wisc.edu/condor/.

social ability can be exploited to plan coordination strategies for an efficient utilization of Grid resources.
– Scheduling can be addressed at application level as Agents execution can be controlled by the server that provides the context inside which they will be hosted. Furthermore persistence and migration can be exploited to suspend, migrate and resume their execution according to defined policies.

Of course many challenges should be addressed in order to grant an effective exploitation of these new facilities. Furthermore common issues of distributed system programming become more critical when applications are based on code mobility.

– It should be granted that Grid programmers do not need to deal necessary with mobile agents technology. It should be possible to reuse original applications without rewriting them. Java cannot be the only language supported because performance could be affected if it was used to implement high performance applications. Besides execution must not be bounded to agent enabled nodes. We do not want to loose the independency of applications from any programming language and the compliance with any Grid technology assured by other platforms.
– *Security* is a main issue when we deal with mobile agents [35]. The hosting node is not aware about the time and from where the agent will come as is pushed and not pulled. We need to take in account the protection of servers against agents; the protection of agents against other agents; the protection of agents against servers; the protection against the network outside the agents domain.
– *Reliability* is a critical feature as the control of distributed nodes is not centralized in Grid architectures. Mobile agents are especially vulnerable to the faults of a distributed system as they autonomously migrate across the network. A failure of the agent server, of the node that is hosting it or the network during the transmission could cause the loss of the agent itself. Not only the agent is lost but also all the results of the computation carried out till the failure event.
– *Network latency*. As Internet is an asynchronous system, the owner could not know if the agent is lost or it is late because of the network and system delay.

4. MAGDA (Mobile Agent based Grid Architecture): A Layered Model

In this section we present the design and partial implementation of an agent based framework for Grid computing: MAGDA – Mobile Agents based Grid Architecture [31]. It is conceived in

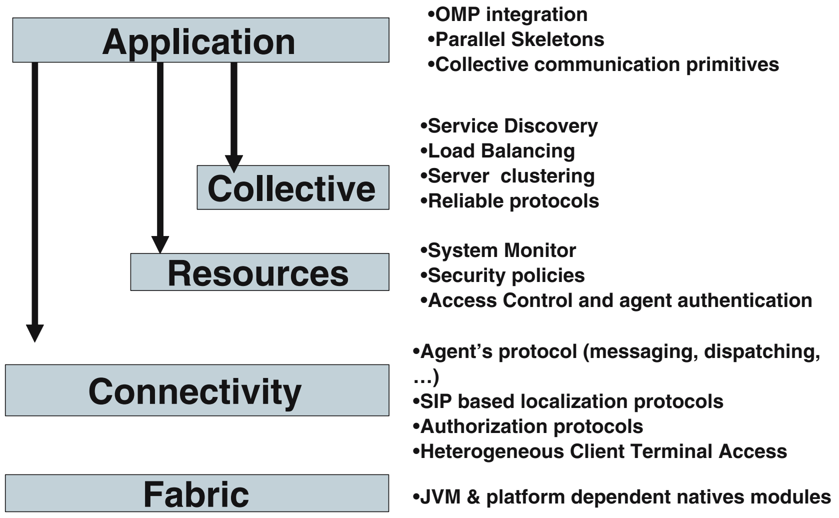
Application
•OMP integration
•Parallel Skeletons
•Collective communication primitives
Collective
•Service Discovery
•Load Balancing
•Server clustering
•Reliable protocols
Resources
•System Monitor
•Security policies
•Access Control and agent authentication
Connectivity
•Agent's protocol (messaging, dispatching, …)
•SIP based localization protocols
•Authorization protocols
•Heterogeneous Client Terminal Access
Fabric
•JVM & platform dependent natives modules

Figure 2 The layered model of MAGDA.

order to provide secure access to a wide class of services in a distributed heterogeneous system, geographically distributed. MAGDA is a layered architecture, which strictly adheres to the Layered Grid Model [12]. In this way any of the several components of MAGDA can be implemented or integrated in any Grid framework, complaint to the Layered Model, by utilizing services provided by components at lower layers, and providing services at upper layers. Figure 2 shows a listing of MAGDA components and services at each Layer of the Grid model. In the following we provide a description of MAGDA components at each layer, starting from lower layers.

4.1. Fabric Layer

At this layer most of all functionalities are provided by the Java Virtual Machine that offers an abstraction of the native system. Where needed, executable programs or native libraries are executed by the Java Runtime System or the Java Native interface. Java is the best choice to support agent mobility across a wide number of heterogeneous architecture because of its portability and thanks to a number of built in mechanisms such as serialization, dynamic class loading, late binding and security isolation. Unfortunately using Java performance of time consuming applications could be affected, however a big benefit is gained in scalability of agent based applications and agent based architecture.

4.2. Connectivity Layer

Here we define protocols which allow the low-level interaction among the components of our Grid architecture. In MAGDA agents exploit the basic communication protocols defined within the Aglet workbench. Further we extended some of them in order to provide localization and security.

4.2.1. *Agent Transfer Protocols*

In order to dispatch agents from a host to another or to support the communication among agents each Mobile Agent platform define its own protocols. The FIPA standard has formally defined the Agent Communication Language (ACL) in order to allow the agents, which execute on platform developed by different parties, to understand one with another. However each implementation can choose to implement different transport protocol to transfer the data across the network. The IBM Aglets toolkit, upon which MAGDA is implemented defines the Agent Transfer Protocol (ATP) to migrate agent and to dispatch messages which is an extension of the HTTP protocol. It also supports http tunnelling to transfer the data beyond a firewall.

4.2.2. *SIP Based Localization Protocols*

While mobile users or agents travel across the network, they always get new addresses and are forced to use different communication protocols. It has been designed and developed a session protocol that allows a connection between two agent servers, and that is transparent with respect to their effective location. To achieve this goal we extended the agent server with a connection manager that implements a SIP compliant protocol [29]. The SIP (Session Initiation Protocol) is a standard for the session creation and management on a data network. It allows the localization of the user and the creation of a control channel that supports the dynamic reconfiguration of the connection at the application level. A SIP register collects the users' registration requests and assigns them an identifier. It is able to localize and to redirect the user's requests to the receiver when a session has to be opened, reconfigured or closed. This service allows the applications to register and update their address, in such a way that the developers and users are able to localize the resource by its identifier. The SIP protocol has been exploited both for server clustering, in the way that will be described at Collective level and for transparent agent localization. Each agent, that needs to be localized by a SIP call, is able to update its new location in a SIP register after it has been dispatched from a host to another. In our implementation the agent server send and accept SIP messages on behalf of its local agents and redirect the body to the right receivers.

4.2.3. *Authorization Protocols*

An Agent needs to be protected from hosts which try to steal or manipulate confidential data carried by the agent itself. We can classify agents security problems according the following criteria:

- Protection of host systems and networks from malicious agents;
- Protection of agents from malicious hosts.

When we deal with mobile agents we have to trust dynamic objects and not static code. Usually each Mobile Agent is an executing thread inside the agent server application. It is characterized by its code, but also by its origin (the place where the execution started), its owner (who started its execution), its execution status, its name (a unique identifier) and eventually its itinerary (the servers he visited). We employ digital signature technology to certify the authenticity of these dynamic properties (not only the code as it is usually done with Java Applets), which change every time the agent migrates from a host to another. Such properties are signed together with the state of the agent by the hosting server just before leaving. Many Mobile Agent Systems address the security issues by employing proprietary mechanisms to protect agents inside their platforms, and adopt standard techniques to protect and trust agents toward any other application. In order to develop a security infrastructure for Mobile Agent systems we need to provide some basic security mechanisms. We developed a component that support the migration by providing it with an authentication mechanism based on digital signature technology. We sign the stream of bytes composed of the state and the code of the agent. The code is enclosed in a Java archive (JAR file),which is signed one time by the developer, before the code has been deployed. The dynamic properties, which characterize the agent instance, and the agent state are signed by the hosting server just before leaving. The signature process generates a standard PKCS#7 file [23]. The PKCS#7 is composed of the plain data, its signed digest and the public digital certificate (which contains the public key among other information) of the agent owner, that is used to authenticate the user and to process its authorization profile. At destination the server is able to verify the signature and to start the agent in case of success. Each agent could be authenticated and authorized by verifying the signature of its byte stream in order to access the required resources and to act on behalf of its owner. A secure dispatching method is provided together with the original non-secure one. The private key of the user is stored in a smart-card and a set of Java API allows to access it when the secure dispatching is selected. The set of developed APIs for the secure dispatch of agents can be used to sign and authenticate communications with other agents or with applications.

4.2.4. *Web Service Based Access and Heterogeneous Client Terminal Access*

In order to provide the interaction between our mobile agent system and any kind of application we are able to deploy the services provided by our Agent Server such as any other Web Service. We defined and implemented a Web Service interface that enables the requestor to create, move, communicate with the agents by the SOAP protocol. We developed an application that implements a wrapper able to receive SOAP requests and to forward them by invoking the Agent API. When the service starts an Aglet Server runs inside the Tomcat Container hosting agents and providing a SOAP bridge that allows any kind of application to interact with them. As the JakartaTomcat application server is unable to handle the ATP (Agent Transfer Protocol) messages it has been extended it in order to support the execution of our application. As handheld devices can exploit just limited resources to access distributed services we are also able to provide a web interface to communicate with agents which execute on MAGDA [29]. An agent server is able to receive an HTTP request from any browsers and to forward it to the addressed agent. The agent is able to process the incoming request as any other messages and can reply providing a content suited for the its client. Each administrative domain should be allowed to define its own requirements for authentication and authorization. A local policy needs to be defined in order to specify who is granted to access the selected resource.

4.2.5. *System Monitoring*

This component had been designed and is currently being developed. In order to support the upper layers in the resource allocation each Grid resource is provided with a parameterized system monitor engine. The MAGDA framework provides developers with some primitives which allow a dynamic monitoring of system parameters such as the CPU and memory utilization. A further significant performance metric is the available bandwidth of the network; it is useful in order to compute the cost of transferring the data and the code from a host to another. A set of Java API has been designed to collect periodically these parameters values by the servers which run on identified operative systems such as Unix or Windows. The exported interface is defined at this layer but its implementation depends on the specific resource and is defined at the Fabric layer. In Figure 3 the *System Monitor* is a component implemented by a thread to periodically collect parameters which characterize the major system activities and resources. The values of interest are stored in a table that can be queried by a *Database Manager*. The *Information Agent* is activated on an information request received as a message. Programmers can exploit this knowledge to optimize resource allocation or to implement effective load balancing strategies. On the other side, when new kind of information needed, a dedicated travelling agent could visit the servers to collect other significant system parameters which are not provided. It could be done by executing specific routines carried by agents on remote hosts. In this case the agents' code is Java and can be executed everywhere, but the we could need to develop different algorithms (shaped according to the specific architecture) to evaluate the status of the target machine.

4.2.6. *Resource Consumption*

In a Grid environment we must avoid that an incoming application is able to reserve and waste a shared resource. Once the agent has been allowed to access a resource, controlling resource consumption is not a trivial task when we deal with Java and mobile agents architectures. Even if we can manage a Java application as any other job scheduled on a Grid nodes, however we have

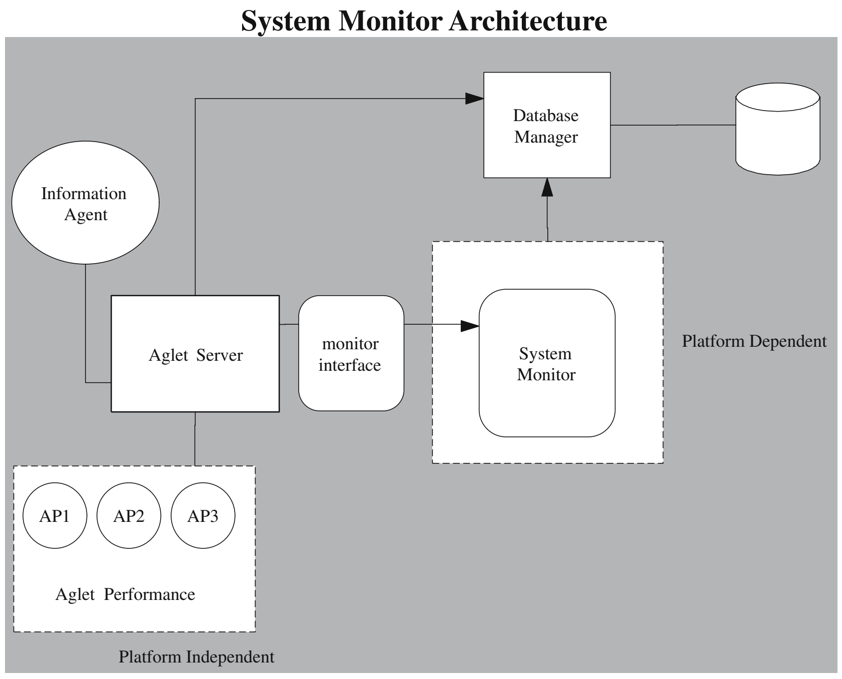

Figure 3 The system monitoring architecture.

not the visibility about the different agents which are running inside the virtual machine. We can control an agent based application defining some parameters which affect the agents' life span. We mean that we can limit the number of hops that an agent is allowed to do (the number of migrations from a host to another), the time life, the minimum time between a suspension and the following resumption, the number of clones and so on. Some mobile agents platforms support this kind of mechanisms which can be used to define and enforce the utilization policy of a shared resource in a Grid environment. Actually we still miss this feature in our architecture as it is not supported by the Aglets workbench that has been used to build a prototypal architecture of MAGDA.

4.2.7. *Security Policies, Access Control and Agent Authentication*

The administrator is able to fix a policy according to which its resource is shared. The policy must be published; this allows each application and user to know what are the constraints on the utilization of the discovered resources. A Security manager, which is able to authenticate the agent code and to process its authorizations, is embedded in the IBM Aglet platform. The Aglet Security Manager processes an authorization policy to grant permissions to the trusted and untrusted agents as a browser with applets. However different processes could share the same code but they could be owned by different users or inherit different permissions. The agent server has been provided with a facility that uses smart card based digital signature to authenticate the stream of data migrated from a host to another.

4.3. Collective Layer

At this layer we implement different sharing behaviors of collective resources. Some examples of services we deal with at this level are resource discovering, clustering and system reliability.

4.3.1. *Load Balancing*

In the Grid the user is just the owner of its application which is hosted by shared resources. The user requests the allocation of resources for its application and can ask for a desired performance profile. The provided resource allocation system, exploiting the services offered by the underlying levels, assigns to the application those available resources that best suit the required performance profile. The goal of the policy chosen to administrate each resource can be the performance optimization, the bandwidth, the throughput or the system utilization. We need to grant coherence between the administration criteria of the shared resources and the application requirements. For example, an application who asks for best performance, after it reserved a hosting node, it must not be scheduled according to a different policy. In order to get the coherence in resource allocation we introduce two proposal approaches.

– According to the first one the application is dispatched together with a resource allocation request and each destination server takes care of satisfying the attached requirements. This choice grants the best QoS to the user, but it builds al lot of constraints on the administration of each shared resource.
– According to the second approach each node must publish the defined policy in order to allow, at collective layer, a finer selection among the available resources before to distribute the application.

Of course to support a dynamical management of resources and its allocation, based on agent cloning and dispatching, we have to provide each node with a set of services at lower level. Some of them have been discussed before: A system Monitor, an information collector, a local resource manager. Even if this component is currently being developed, at application level a load balancing facilities are already operational and have been exploited in the case study described in Section 5.

4.3.2. *Service Discovery*

We are designing and have implemented a service discovery functionality for the MAGDA platform, mainly exploiting web services technology [28]. Within MAGDA framework, resources and services are characterized by mobility features. In fact in our architecture a resource is defined as

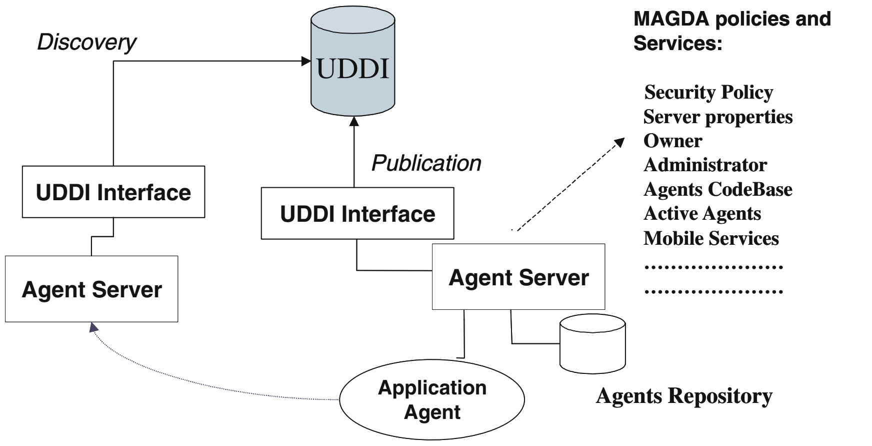

Figure 4 Resources and application discovery.

a node able to host a mobile agent. A service is an application server (that can host an agent) or a mobile agent. Both users and applications must be able to get the visibility of the Grid in order to ask for its resources. They need to know what are the available resource, and for each of them, their exported services and access policies. The user is able to discover available services (running agents) or to start its own services downloading the agent code or asking for agent's creations. Also the agents are able to discover new hosting node, in order to explore the network, to migrate to less busy machines, or to look for other applications.

Our design follows the web services approach and reuses its technology. Each agent server is a services provider and must publish its address and services in a UDDI register. These references are bind to specific applications or data. As shown in Figure 4 the platform is able to access an UDDI register by a set of extended UDDI APIs. At least the publication should provide the following items:

- The features of the shared resource: Amount of memory, CPU power, number of nodes, bandwidth of the network, operative system, available compilers, etc.
- Access and authorization policies
- Owner and administrator's reference
- System services: System monitoring, code based repository, etc.
- Application services: Running agents

In our prototypal implementation the published services are references to mobile agents. The search is available inside an application by the developed APIs, or by an extension of the Tahiti graphical interface. The services are accessible also by a Web Service interface.

4.3.3. *Server Clustering*

In a Grid environment, a relevant feature that should be provided is the clustering of the shared resources in order to manage and exploit them how better it is possible. Aims of clustering are:

- the global visibility of the network configuration that means both the topology of the system and the features of links and nodes;
- the design of efficient communication patterns for multicast and broadcast messaging;
- the support for agents to discover new agent servers where to migrate and to be hosted;
- the possibility to look for other agents executing on a clustered nodes.

We have designed a software protocol which is employed to organize the agent servers with a regular binary tree topology. The cluster of agent servers is dynamically updated by means of enroll and disjoin procedures. The requests are submitted to an elected master server that communicates the changes to the involved members. Each new computing node, joining the MAGDA cluster, checks the presence of a master node in the network. If the master node is not reachable it promotes itself as master node, otherwise if a master node is already up it will be contacted and will take care of providing the new node with a new identity in the topology and with the reference to its neighbors. Each node, which is already part of the tree, is also notified by the server node of the changes in the structure. Each node in the tree is able to directly reach its neighbors or to ask them for the other destinations in the tree. The knowledge of the topology provides an agent with the possibility to explore the network. Further we were able to augment the framework with collective communication primitives. Each server is able to forward broadcast and multicast messages across the web of active nodes exploiting ad hoc communication paths upon the defined topology. Mobile nodes as laptop and handheld devices can be moved from a network to another getting different addresses. The roaming of a mobile user could require many changes of the system configuration affecting performance with a relevant overhead and reliability. The SIP protocol, widely used in Voice Over IP applications, supports the localization of the user who is roaming in the network by a SIP identifier and allows to open and manage sessions. The localization of the clustered agent servers has been performed in our architecture by the SIP compliant protocol provided at connective layer. Prototypal implementations of the described architecture has been developed and then it reliability has been improved by the protocols described in the next section.

4.3.4. *Reliable Protocols*

In a distributed system we have to deal with the management of unexpected events such as a fault or a degraded behavior of the system. In order to manage such a kind of occurrences we introduce different protocols to ensure both the reliability to the applications and a dynamic reconfiguration of the available resources in the Grid platform. As the Grid is built on a distributed system and no one has a complete control of it, it is a main issue to grant reliability. In order to manage a fault occurrence or a degraded behavior of the system we need to:

- detect the fault
- notify it to the interested components
- reconfigure the system by cutting off the faulty component
- notify the changes in the configuration
- recover data and applications, when it is possible

We introduce here two reconfiguration protocols which are designed in order to provide reliability to the applications and to the Grid configuration. The first one is based on the well known leader election protocol, which is used to coordinate asynchronous processes. It is conceived to grant availability and the execution recovery to the applications. The fault must not be able to interrupt the service for more than a minimum amount of time. The application should be able to recover a correct instance of its execution state, saved before the fault, and to restore the service. We suppose to have an agent that is carrying on a sequential task. In order to grant the promised reliability the system provides to create some clones of the first agent. The clones are dispatched to different servers where they must be ready to replace an agent if it faults. Through a cyclic polling each agent asks its clones for reply, in order to detect any fault. When a fault occurs among the agents a leader is elected, whose task is to replace the faulty clones. The execution state is forwarded from the master clone to the others when it needs. The reliability of the application increases with the number of clones, however

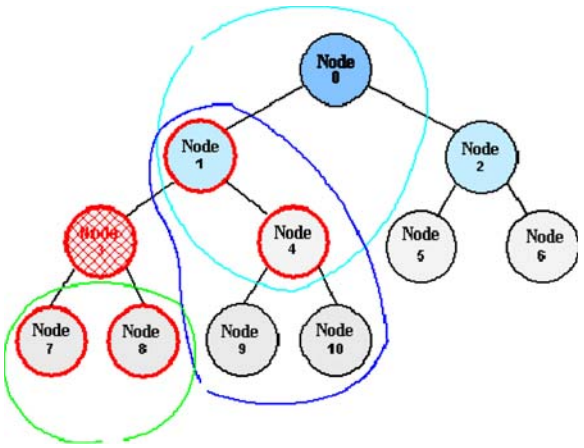

Figure 5 A reliable protocol for resource clustering reconfiguration.

the performance of the application and the load of the system grow. About the reliability of the Grid architecture we designed and implemented a protocol that allows to reconfigure the shape of the clustered resources when a node fails. The classic leader election protocol is not suited in this case because it does not provide scalability to the architecture. Performance is lost too quickly when the number of nodes increases. In our extension each node acts just a monitoring of its neighbors as shown in Figure 5. The generic node belongs to two disjoint groups. In Figure 5 the node 3 belongs to the groups {1,4,9,10} and {7,8}. If it was faulty, the components of each group should agree about the occurrence of the failure. Only one node per group is elected to ask the master node (node 0) for the substitution of the faulty node. If the mater receives both the alerts from the elected nodes it substitutes in the topology the faulty node with the last registered one. The changes are communicated to all the interested nodes. The leaves and the root node belong to a single group so a different strategy is chosen.

4.4. Application Layer

The higher layer in Grid architecture consists of end-user applications developed relying on services defined at any lower layer. The main issue, at this level, is to make easy and efficient the task of an application developer. In MAGDA, we support parallel programming by the mobile agent paradigm with a set of Java packages that:

– help the building of distributed applications through the use of different predefined parallel skeletons
– allow an effective utilization of the available Grid resources by means of dynamic load balancing mechanisms customized to the specific application
– provides collective communication primitive for agent based applications

Some examples of services at this layer are described in the following.

4.4.1. *OpenMP Integration*

In order to program hierarchical distributed-shared memory multiprocessor architectures, in particular heterogeneous clusters of SMPs (and uniprocessor) nodes, mobile agents and high-level shared memory programming paradigms can be coupled in order to express a hierarchical (two-level) parallelism: An external distributed memory level, and an internal shared memory one. The agents paradigm implements the external distributed memory parallelism; internal shared memory parallelism can be achieved within each agent's execution, and can be expressed through OpenMP directives. We have obtained such a coupling through the integration of MAGDA with OpenMP compiler technology [26]. In particular we have utilized the JOMP Compiler, developed by the EPCC research center of the University of Edinburgh [24]. It is an OpenMP compiler for the Java language, augmented with OpenMP like directives and methods, proposed by the authors. It provides an implementation of most OpenMP directives for parallel regions, synchronization and mutual exclusion among threads. It provides a runtime library, in the form of a Java class library. The integrated use of JOMP within the Aglet workbench is straightforward: The JOMP compiler needs to be used just as a pre-processor for the Aglet Workbench, for the Java agent classes whose methods contains OMP annotations. The case study presented in Section 5 exploit this integration.

4.4.2. *Algorithmic Skeleton*

Parallel programming effort can be reduced by using high-level constructs such as algorithmic skeletons. We provide a skeleton-based parallel programming environment, based on specialization of Algorithmic Skeleton Java interfaces and classes to support development and execution of mobile agent based distributed applications. It is operational and has been described in [25]. The user can thus develop a parallel, mobile agent based application by simply specializing a given set of classes and methods and using a set of added functionalities. As a matter of fact, developing distributed applications using the mobile agent model remains a non-trivial task because the mobility feature introduces additional difficulties in designing coordination, synchronization and communications among the different workers. In practice, many applications share the same communication and synchronization structure, independently from the application-specific computations. In addiction such approach allows to reuse a great deal of the sequential code, when available. A predefined algorithmic skeleton allows to follow the sequential programming model by filling some methods, classes and interfaces and to hide the difficulties involved by an explicit parallel programming paradigm. So, especially when the starting point is an available sequential code, using the concept of the algorithmic skeletons that is separating the communication/synchronization structure from the application-dependent functions, can ease the programming task, and improve the mapping for performance o parallel systems. Finally exploiting the peculiar features of both Object Oriented and mobile agents programming models, the proposed skeletons-like approach enables to program distributed applications. By means of the provided skeletons interfaces the programmer is able to implement his own application by specializing an assigned structure and utilizing the set of functionalities that the mobile agents framework offers. Two algorithmic skeletons involving the Farm-like programming paradigm and the Divide and Conquer-like programming paradigm, respectively, have been implemented and tested. In the Processor Farm programming paradigm the master process creates a number of slaves and assigns some work to every one of them. The slaves compute their work and return the results to the master. Task Queue is the most general Farm-like skeleton; every slave may produce new work to be performed by itself or by other slaves. The second algorithmic skeleton we have implemented belongs to Divide and Conquer-like skeleton class, but not to the highest abstraction level. It is an example of Tree computation algorithmic skeleton. It solves the initial problem dividing it in several sub-problems assigned to different agent workers. The data flow down from the root into the leaves and the solutions flow back toward the root. We have chosen to implement a binomial algorithm to build our tree, so its shape and the results recombination procedure are consequentially determined.

4.4.3. *Collective Communication Primitives*

In a highly dynamic computing platform that represents the target computing environment of most mobile agent based applications, it becomes essential for the user to benefit of some collective mechanisms of communication and synchronization. We provide a set of Java API that allows collective communication among mobile agents which are spread across the network. We were able to augment the framework with collective communication primitives such as broadcast and multicast, and some collective synchronization operations that, for example, allow to compute and distribute a maximum or minimum value among all the agents involved. The agent servers are clustered according to the protocols defined at the collective layer. Communication patterns can be built on top of the cluster topology. As an example, in order to send a remote multicast, an agent must exploit the ‘remote multicast’ primitive and provide the message with a label. An agent who needs to receive multicast messages has to subscribe itself to the reception of a multicast message with the same specific label. The message is broadcasted across the cluster along a sink tree, and on each server the message is forwarded to all the subscribed agents. We implemented and test different implementations which exploit agent migration to dispatch messages across a network through different patterns.

4.4.4. *A Dynamic Workload Balancing Facility for Application Programming*

Load balancing can be implemented at different layers of the introduced architecture, according to the issue that should be addressed. At application layer we intend with Workload balancing the possibility to distribute the application workload among the worker agents. This approach requires to know the semantic of the application. We provide a service for dynamic workload balancing that can be easily customizable to any user application developed within the Workbench. It is fully operational and it was described in detail in

[27]. The framework is composed of a coordinator agent that controls the load balancing and an Aglet super-class support. The coordinator agent communicates, by message passing, only with the Aglet super-class. In order to configure the support, the user class must override some methods inherited by the super-class and set some parameters. The implementation of these functions depends on the specific user application. The coordinator manages a first list of registered workers and a second list of available free hosts. When the user Agent's execution starts, the coordinator is created and executes a polling among the working agents registered with it. It is done in order to know the state of the workers' computation, which is the percentage of computation performed, with respect to the amount assigned to it. The registered agent's references are stored in a vector, ordered according to their computation state; the ordering of this vector thus represents the relative speed of each worker with respect to the others. It also gives a representation of the state of the computation. A load unbalance event occurs when: 1) A worker ends the computation assigned to it, and becomes idle; 2) the slowest worker has completed a percentage of the computation assigned to it which is far below the percentage completed by the fastest worker (determined by a fixed threshold on the difference of the two percentage). In this case the coordinator ask the most loaded worker to split its workload that will be assigned to the less loaded one.

5. A Case Study

We provide here an example of parallel application developed upon the MAGDA framework. We describe how some MAGDA facilities have been exploited and provide some considerations about performance results. We show, through a case-study, how to yield a hierarchically distributed-shared memory parallel version of a sequential algorithm, without completely rethinking its structure, reusing a great deal of the sequential code and trying to exploit the heterogeneity of the target computing architecture. We exploit collective communication, the OpenMP and the balancing facilities of the MAGDA framework. The chosen application solves the well-known N-body problem [32]. The mobile agents and high-level shared memory programming paradigms have been coupled in order to express a hierarchical (two-level) parallelism: An external distributed memory level, and an internal shared memory one. The interacting agents' execution model perform the external distributed memory parallelism; internal shared memory parallelism can be achieved within each agent's execution, and can be expressed through OpenMP directives. The chosen case-study is a sequential algorithm that solves the N-body problem by computing, during a fixed time interval, the positions of N bodies moving under their mutual attraction. The algorithm is based on a simple approximation: The force on each particle is computed by agglomerating distant particles into groups and using their total mass and center of mass as a single particle. The program repeats, during a fixed time interval, three main steps: To build an octal tree (octree) whose nodes represents groups of nearby bodies; to compute the forces aging on each particle through a visit in the octree; to update the velocities and the positions of the N particles. The parallelization of the code relative to the construction of the octree, needs that the reading of the input data and the production of the first level in the octree is carried out by the first running agent. As soon as the nodes in the current level of the octree exceeds in number the computing nodes, the first agent clones itself and dispatches its clones to each host making up the computing environment. Every agent, are responsible for the building of the subtrees assigned to it, an operation that can be carried out in parallel. At the end of this stage every agent has filled up a slice of the complete octree data structure and can send a multicast message to all the other agents so that each of them is able to get a complete copy of the octree. The computation stage of the force acting on a single particle, as well as the update of its velocity and position results to be completely independent. Each of the N particles, in fact, computes the force aging on it by traversing separately the octree, just as the updating phase of the particles velocities and the positions requires the same computation on different data. So, a parallelization can be obtained by simply

Table 1 Connection links.

Node	Max speed (Mbits/s)	Mean distance (km)
Portici	18.0	290
Frascati	18.0	413
Trisaia	4.0	327
Brindisi	2.0	380

distributing the particles among the agents. In this case it's immediate to write a multi-thread version of forces computation stage by inserting the BALANCED FOR directive inside the agent code, in order to exploit the availability of nodes with multiple processors. The benefits of the dynamic workload balancing infrastructure available in our programming framework is exploited to distribute the loop iteration chunks among the agents. A coordinator agent assigns chunks to each worker agent, which either starts the computation of the loop, or asks for new workload. Each worker agent performs the computation corresponding to the assigned chunk, possibly in parallel through multiple threads. Finally, at the end of their work, the agents, by means of a new all-to-all multicast message provided by a MAGDA facility, obtain an updated copy of the particles velocities and positions so that the next iteration step of the algorithm can start. It is worth while underlining that the original sequential routines with the appropriate input parameters can be completely reused becoming different methods of the agent code.

5.1. Experimental Results

Experiments we describe in this section have been carried out on a production Grid owned by ENEA (National Agency for Energy and Environment). The Grid subsystem we used for our experiments is composed of four four-way processors SMP nodes. All computing nodes were equipped with four Xeon 3 GHz processors and 8 GB RAM, located in four towns of the middle and the south of Italy (Frascati, Portici, Trisaia, Brindisi) and connected by different bandwidth connection links. In Table 1 we report the mean distance between a node and the other and the bandwidth of the different links. The force computing phase is the most CPU-consuming portion of the program and at the same time the most significant to our purposes. As it was explained above we expect to have he maximum speed-up for the computation of the forces acting on each particle were as both the OpenMP extension and the parallelization by the agents are exploited. In Figure 6 we can compare the ideal speedup with the real one and with the one obtained just considering the force computation. We provide here just the performance figures obtained when, on the distributed system, we executed one agent for each node. On the *x*-axis we reported the maximum number of parallel threads spawned for each experiment, but we need to clarify that just the last measure exploits all the four nodes of the Grid. In particular on the last experiment we added the ones which were connected to the network by more narrow-

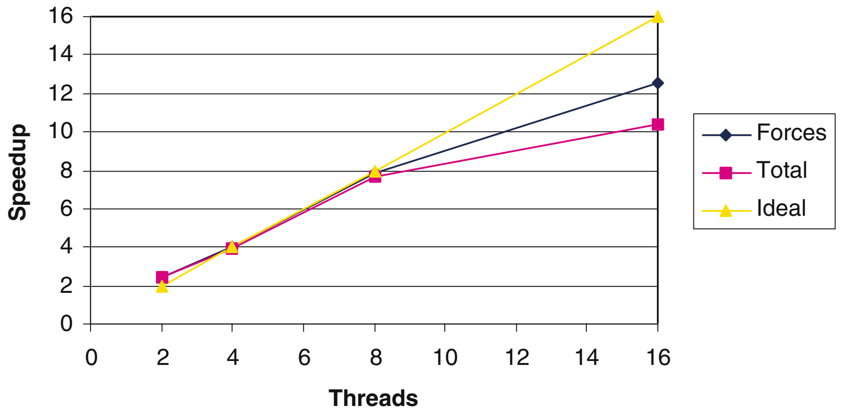

Figure 6 Speed-up of the N-body application on Linux distributed system, two of four nodes, four processors.

band links. We must consider that server have been shared with other users and our application never get exclusive access to the resources. Besides, above all for Trisaia and Brindisi the bandwidth used is always near saturation. This means that it is difficult to get any significant consideration without an extensive analysis of a wide amount of experimental results performed in different condition. Trying to overcome the unbalance due to the irregularity of the problem and to the unpredictable dynamic condition of nodes and traffic network we exploited the balancing facility intra and inter each node. As we can see the performance figure is very promising as the speedup increase well, even if we did not care to optimize the application that was originally conceived to run on a local cluster. To summarize we can say that, in particular, for low number of threads the speed-up of the balanced version is near to the ideal speed-up value. We can conclude that agents, under the tested conditions, exploiting some MAGDA facilities as dynamic workload balancing service together with the multithreaded execution promise to be effective to obtain a good scalability on distribute computing architecture.

6. Conclusions

MAGDA, an example of Mobile Agent based platform for Grid programming we have designed and are developing has been presented. In Section 5 a case study application of MAGDA, on a production Grid System, has been presented. The interested reader can find in the following references additional details on our gained experience with use of Mobile Agents technology in Grid computing and with MAGDA features in particular: Dynamic load balancing facilities and their exploitation in distributed applications are described in [27]; an integration of OpenMP compiler technology with a mobile agents environment to develop and execute hierarchically distributed-shared memory applications on Grid of SMP nodes are described in [24]; design, implementation and utilization of a set of API that provides high level constructs to support the development and the execution of parallel applications have been presented in [25]; a comparison between the mobile agents and the distributed object programming paradigms is provided in [30]. The ever increasing adoption and improvements of Mobile Agents technology, the Java performance improvements, the widespread adoption of the FIPA standard for agents interoperability, encourage us to continue and extend the design and implementation work performed until now, in particular towards the integration of MAGDA functionalities within a currently available open source Grid framework.

Acknowledgments Experiments have been carried out at: Associazione Euratom-ENEA sulla Fusione, Frascati, Rome. We are particularly thankful to Sergio Briguglio, Gregorio Vlad, and Giovanni Bracco and Silvio Migliori from Servizio Centralizzato Informatica e Reti.

References

1. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Karonis, N., Toonen, B., Foster, I.: J. Parallel Distrib. Comput. (2003)
2. Globus: A Metacomputing Infrastructure Toolkit. Foster, I., Kesselman, C.: Int. J. Supercomput. Appl. **11**(2), 115–128 (1997). Provides an overview of the Globus project and toolkit
3. Nick, J.M., Tuecke, S., Foster, I., Kesselman, C.: The physiology of the Grid: An open Grid services architecture for distributed systems integration. Technical report, http://www.globus.org/research/papers/ogsa.pdf (2002)
4. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Trans. Softw. Eng. **24**(5), (May 1998)
5. Pham, V.A., Karmouch, A.: Mobile software agents: An overview. IEEE Commun. Mag. **36**(7), 26–37 (July 1998)
6. Lange, D., Oshima, M.: Seven good reasons for Mobile Agents. Commun. ACM **42**(3), (March 1999)
7. Baratloo, A., Karaul, M., Kedem, Z., Wycko, P.: Charlotte: Metacomputing on the web. In: 9th International Conference on Parallel and Distributed Computing systems, Dijon, France, 1996
8. Tveit, A.: jfipa – An architecture for agent-based Grid computing. In: AISB'02 Convention, Symposium on AI and Grid Computing, London, United Kingdom, 2001
9. Niranyan, S., Groth, P.T., Bradshaw, J.M.: While you're away: A system for load-balancing and resource based on mobile agents. In: Buyya, R. (ed.) 1st IEEE International Conference on Cluster Computing and the Grid, Brisbane, Australia. IEEE Computer Society (2001)

10. Cao, J., Kerbyson, D.J., Graham, R.N.: High performance services discovery in large-scale multi-agent an mobile-agent systems. Int. J. Softw. Eng. Knowl. Eng. **2**(5), 621–641 (2001)
11. Bruneo, D., Guarnera, M., Zaia, A., Puliafito, A.: A Gridbased architecture for multimedia services management. In: Annual CrossGrid Project Workshop, 1st European Across Grids Conference, 2003
12. Foster, I.: The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science 2150* (2001)
13. Binder, W., Di Marzo Serugendo, G., Hulaas, J.: Towards a secure and efficient model for Grid computing using mobile code. In: 8th ECOOP Workshop on Mobile Object Systems, Agent Applications and New Frontiers, Malaga, Spain, June 10, 2002
14. Fukuda, M., Tanaka, Y., Suzuki, N., Bic, L.F., Kobayashi, S.: A mobile-agent-based PC Grid. In: Proc. of the 5th Annual Int'l Workshop on Active Middleware Services – AMS2003, Seattle, Washington, pp. 142–150, June 25, 2003
15. Tomarchio, O., Vita, L.: On the use of mobile code technology for monitoring Grid system. CCGRID 450–455 (2001)
16. Di Martino, B., Rana, O.F.: Grid performance and resource management using mobile agents. In: Getov, V., Gerndt, M., Hoisie, A., Malony, A., Miller, B. (eds.) Performance Analysis and Grid Computing. Kluwer (November 2003)
17. Kuang, H., Bic, L., Dillencourt, M.B.: Iterative Grid-based computing using mobile agents. ICPP 109–117 (2002)
18. Chakravarti, A.J., Baumgartner, G., Lauria, M.: The organic Grid: Self-organizing computation on a peer-to-peer network. ICAC 96–103 (2004)
19. Ma, T., Li, S.: An instance-oriented security mechanism in Grid-based mobile agent system. IEEE International Conference on Cluster Computing 492–495 (2003)
20. Kurkovsky, S., Bhagyavati: Modeling a computational Grid of mobile devices as a multi-agent system. In: Proceedings of The 2003 International Conference on Artificial Intelligence (IC-AI'03), Las Vegas, Nevada, (June 2003)
21. Hingne, V., Joshi, A., Finin, T.W., Kargupta, H., Houstis, E.N.: Towards a pervasive Grid. IPDPS 207 (2003)
22. Hingne, V., Joshi, A., Finin, T., Kargupta, H., Houstis, E.: Towards a pervasive Grid, Parallel and Distributed Processing Symposium (IPDPS)2003, 22–26 April 2003, IEEE CS Press, 2003, ISBN: 0-7695-1926-1
23. Labs, P.R.: Pkcs7: Cryptographic message syntax standard, "http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/index.html", 1997. Printed by RSA
24. Bull, M., Westhead, M., Kambutes, M., Obdrzalek, J.: Towards OpenMP for Java. In: Proc. of 2nd European Workshop on OpenMP – EWOMP'2000, Edinmburg (UK), (14–15 September 2000)
25. Aversa, R., Di Martino, B., Mazzocca, N., Venticinque, S.: Mobile agent programming for clusters with parallel skeletons. In: Palma, J.M.L.M., Dongarra, J., Hernandez, V., Sousa, A.A. (eds.) VECPAR'2002. 5th International Conference on High Performance Computing in Computational Sciences 2002. Selected Papers and Invited Talks, Lecture Notes in computer Science, vol. 2565, pp. 614–627. Springer, Berlin Heidelberg New York (2003). (ISBN 3-540-00852-7)
26. Aversa, R., Di Martino, B., Mazzocca, N., Rak, M., Venticinque, S.: Integration of mobile agents and OpenMP for programming clusters of shared memory processors: A case study. In: Proc. of EWOMP (European Workshop on OpenMP), 2001, Barcelona, Spain, (8–12 Sept. 2001)
27. Aversa, R., Di Martino, B., Mazzocca, N., Venticinque, S.: Mobile agents for distribute and dynamically balanced optimization applications. In: Hertzberger, B., et al. (eds.) High-Performance Computing and Networking (Lecture Notes in Computer Science, vol. 2110), pp. 161–170. Springer, Berlin, (2001), (ISBN: 3-540-42293-5)
28. Aversa, R., Di Martino, B., Mazzocca, N., Venticinque, S.: A resource discovery service for a mobile agents based Grid infrastructure. In: Yang, L.Y.,Pan, Y. (eds.) High Performance Scientific and Engineering Computing, pp. 189–197. Kluwer Academic publishers, Boston (2003), (ISBN: 1-4020-7580-4)
29. Aversa, R., Di Martino, B., Mazzocca, N., Venticinque, S.: Terminal-aware Grid resource and service discovery and access based on mobile agents technology, Parallel Distributed and Network based Processing (PDP04), IEEE, 2004, February, 11-13, 2004. A Coruna, Spain, ISBN: 0-7695-2083-9, pp. 40–48
30. Aversa, R., Di Martino, B., Fahringer, T., Venticinque, S.: On the evaluation of the distributed objects and mobile agents programming models for a distributed optimization application. In: Goos, G., Hartmanis, J., Leeuwen, J. (eds.) Applied Parallel Computing (Lecture notes in Computer Science vol. 2367), pp. 233–242. Springer Verlang, Berlin Heidelberg New York (2002), ISBN:3-540-43786-X
31. Aversa, R., Di Martino, B., Mazzocca, N., Venticinque, S.: MAGDA: A software environment for mobile agent based distributed applications. In: Parallel Distributed and Network based Processing (PDP03), Genova, Printed by IEEE Computer Society (2003) ISBN: 0-7695-1875-3, pp: 332–338
32. Grama, A., Kumar, V., Sameh, A.: Scalable parallel formulations of the Barnes–Hut method for *n*-body simulations. Parallel Comput. **24**(5–6), 797–822 (1998)
33. Rana, O.F., Moreau, L.: Issues in building agent-based computational Grids, UK Multi-Agent Systems Workshop, Oxford, (December 2000)
34. Rana, O.F., Walker, D.W.: The agent Grid': Agent based resource integration in problem solving environments, 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, Lausanne, Switzerland, August 2000
35. Nitschke, L., Paprzycki, M., Ren, M.: Mobile agent security – An overview. In: Niedzielska, E., et al. (eds.)

Modern Information Technologies in Management, pp. 600–608. Wroclaw University of Economics (2004)
36. Tianfield, H., Unland, R.: Towards self-organization in multi-agent systems and Grid computing. Multiagent and Grid Systems Journal, IOS Pres **1**(2), 89–95 (2005)
37. Li, Z., Parashar, M., Rudder: An agent-based infrastructure for autonomic composition of Grid applications. Multiagent and Grid Systems Journal, IOS Pres **1**(3), 183–195 (2005)
38. Coddington, P.D., Lu, L., Webb, D., Wendelborn, A.L.: Extensible job managers for Grid computing, ACM proceedings of the twenty-sixth Australasian computer science conference on research and practice in information technology, vol. 16, pp. 151–159. Australian Computer Society, Australia (2003), ISBN:1445-1336
39. Mirtchovski, A., Simmonds, R., Minnich, R.: Plan 9 – An Integrated Approach to Grid Computing, IPDPS2004, 26–30 April 2004, New Mexico, USA, ISBN:0-7695-2132-0
40. Grimshaw, S., Humphrey, M.A., Natrajan, A.: A philosophical and technical comparison of Legion and Globus. IBM J. Res. Dev. **48**(2), (March 2004)
41. Smith, W., Hu, C.: An Execution Service for Grid Computing, NAS Technical Report, (April 2004)
42. Kolano, P.: Facilitating the portability of user applications in Grid environments. In: Proc. of the 4th IFIP Intl. Conf. on Distributed Applications and Interoperable Systems, Paris, France, Nov. 18–21, 2003
43. Jain, A., Shyamasundar, R.K.: Failure detection and membership management in Grid environments. Grid, pp. 44–52, Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), 2004
44. Nwana, H.S.: Software agents: An overview. Knowl. Eng. Rev. **11**, 1–40 (Sep. 1996)