Available online at www.sciencedirect.com

ScienceDirect

Future Generation Computer Systems 24 (2008) 17–29

FGCS FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

MiPeG: A middleware infrastructure for pervasive grids

Antonio Coronato[a], Giuseppe De Pietro[b,*]

[a] SASIT-CNR, Via P. Castellino 111, 80131 Napoli, Italy
[b] ICAR-CNR, Via P. Castellino 111, 80131 Napoli, Italy

Received 16 January 2007; received in revised form 3 April 2007; accepted 6 April 2007
Available online 21 April 2007

Abstract

Grid computing and pervasive computing have affirmed respectively as the paradigm for high performance computing and the paradigm for user-friendly computing. The conjunction of such paradigms are now generating a new one: the Pervasive Grid Computing. This paper presents a middleware for pervasive grid applications. It consists of a set of basic services that aim to enhance classic grid environments with mechanisms for: (i) integrating mobile devices in a pervasive way; (ii) providing context-awareness; (iii) handling mobile users' sessions; and, (iv) distributing and executing user tasks on the devices (even mobile) active in the environment.

Keywords: Grid; Pervasive Computing; Middleware services

1. Introduction

During the last decade, new computing models have emerged and rapidly established themselves. In particular, terms like *Grid Computing*, *Pervasive Computing* and *Utility Computing* have become common usage not only in the scientific and academic world, but also in business fields.

The *Grid Computing* model has demonstrated itself to be an effective way to face very complex problems. The term "The Grid" was primarily introduced by Foster and Kesselman to indicate a distributed computing infrastructure for advanced science and engineering [1]. Successively, it has been extended to denote the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities. As a result, Grids are geographically distributed environments, equipped with shared heterogeneous services and resources accessible by users and applications to solve complex computational problems and to access to big storage spaces.

Differently, the goal for *Pervasive Computing* is the development of environments where highly heterogeneous hardware and software components can seamlessly and spontaneously interoperate, in order to provide a variety of services to users independently of the specific characteristics of the environment and of the client devices [2]. Therefore, mobile devices should come into the environment in a natural way, as their owner moves, and transparently, so that the owner will not have to carry out manual configuration operations to approach the services and the resources, and the environment has to be able to self-adapt and self-configure in order to host incoming mobile devices.

On the other hand, *Utility Computing* aims at providing users with computational power in a transparent manner [7], similarly to the way in which electrical utilities supply power to their customers. In this scenario, computing services are seen as "utilities" that users pay to access to, just as is in the case of electricity, gas, telecommunications and water.

Classic grid applications are neither pervasive, nor able to implement the Utility Computing vision in a completely transparent way. As a matter of fact, classic grids do not provide mobile users with support to access resources and services at all (i.e. classic grids consists of wired, pre-configured, powerful stations). Moreover, whenever a user wants to execute her own application, she typically has to: (i) ask the grid for resources; (ii) allocate tasks; (iii) launch and control executions; (iv) get results; and (v) release resources.

* Corresponding author. Fax: +39 0816139531.
E-mail addresses: coronato.a@na.drr.cnr.it (A. Coronato), depietro.g@na.icar.cnr.it, giuseppe.depietro@na.icar.cnr.it (G. De Pietro).

This practice has several limitations:

1. User-environment interactions are very little transparent;
2. Users have direct control of allocated resources. The user requires (and sometimes locks) resources of the grid;
3. Resources are handled in an insecure and inefficient way. A malicious user could require a larger amount of resources with respect the ones really needed or an inexpert user could underestimate the resources that his application should get assigned.

It is worth noting that even in the case of brokering systems, like Condor [24], which directly handle the set of grid resources and enable users just to submit tasks for execution, several limitations are present. As a matter of fact, Condor doesn't take into mobile resources and mobile users. Such limitations have partly been faced in works like [6], which aim at making Condor services available to mobile users. However, these solutions continue to neglect mobile resources as available (useful) resources for the grid. Differently, our implementation not only enable mobile users to get access to services and resources by means of their mobile devices, but also it integrates mobile devices as active grid resources. As a result, tasks can be launched on mobile resources. In particular, we have performed some experiments in which the entire branches of code have been executed only on mobile resources without loading classic grid resources. On the one hand, this is a way to extract useful computing power from resources that have historically been neglected by classic grids. On the other hand, this way the proposed middleware provides support to easily develop specific applications that require the execution of branches of code by mobile devices or sensor networks.

In this paper, we present a middleware infrastructure able to integrate mobile devices in the grid.

Mobile equipments are integrated either as mechanisms for interfacing the grid or as active resources for the grid itself.

In the case of access mechanisms, every incoming mobile device is integrated in a completely transparent way; that is, the mobile user can access the grid and move within the environment without any manual configuration operation. In particular, the mobile user gets automatic access to services as soon as he enters the environment. In addition, whenever he moves within the environment, his device is automatically located and his context follows him.

On the other hand, in order to integrate mobile devices as active resources for the grid (i.e., resources able to execute tasks as in [3] or to expose services), a light middleware branch must be installed and launched on them. This is a one time configuration operation. After that, whenever the mobile device enters the environment it is automatically recognized and monitored by the environment, which is then able to allocate tasks on such a mobile resource or to make the services carried by it available for the grid.

Finally, the middleware exposes a basic service for the automatic execution of Java tasks according to the *Utility Computing* model. As a matter of fact, every user can submit its own Java tasks that are automatically replicated and allocated for execution on available resources, both wired or wireless. After execution, the environment gathers results and returns them to the user. An application of such a service has been adopted to develop tools for performing parallel software testing. As a matter of fact, when a developer has to test a branch of code, he has to design test cases (i.e. sets of data inputs) and execute the code for every test case. By the way, each execution is independent from every other, and they can be performed independently. The testing service under development will enable a programmer to submit the software element to test and the set of test cases. After that, it will be in charge of executing the software element with respect to each test case. Executions can be performed contemporaneously on different devices. Since testing activities may require the execution of hundreds of test cases, it is clear that such a service will drastically reduce the effort for testing software elements.

The rest of the paper is organized as follows. Section 2 discusses some motivations, related work and contributions. Section 3 describes the high-level architecture of the middleware infrastructure. Section 4 details basic services. In Section 5, a running scenario and some experimental results are presented. Finally, Section 6 concludes the paper.

2. Motivations, related work and contribution

2.1. Motivations and related work

Mobile and wireless devices have not been considered, for a long time, as useful resources by traditional Grid environments. However, considering Metcalfe's law, which claims that usefulness of a network-based system proportionally grows with the square of the number of active nodes, and also considering that mobile devices capabilities have substantially be improved over the time, it can justifiably be stated that mobile and wireless devices are now of interest for the Grid community [4].

In particular, they have to be incorporated into the Grid either as service/resource consumers or as service/resource providers [9]. A very interesting example is the "NEESgrid" [26], which is a grid infrastructure for earthquake forecasts and simulations that have required the integration of cameras, sensor networks and mobile devices for measures in the field.

However, integration is not costless [10]. This is mainly due to the consideration that current Grid middleware infrastructures don't support mobile devices–not only because they have not been devised taking into account pervasive requirements like spontaneity, transparency, context-awareness, pro-activity, and so on, but also for three main reasons: (i) they are still too heavy with respect to mobile and wearable equipments capabilities; (ii) they assume fixed TCP/IP connections and do not deal with wireless networks and other mobile technologies; and, (iii) they typically support only one interaction paradigm, that is SOAP messaging, whereas the Pervasive model requires a variety of mechanisms [13].

Over the last years, some valuable efforts have been made in order to make Grid architectures able to support wireless technologies and mobile devices. In particular, the paradigm of Mobile Grid or Wireless Grid has been proposed [4, 8–10]. More recently, this paradigm has evolved in the Pervasive Grid model [11,12], which again aims at making Grid environments able to integrate mobile devices, but in a pervasive way–that is seamlessly and transparently. In addition to this, the final objective is both to enhance Grid environments with characteristics like context-awareness and pro-activity, which are typically found in Pervasive environments, and to implement the Utility Computing model, which would ease and make more robust the execution of user tasks.

It is worth noting that the effort of exploring possibilities of synergy between the Pervasive Computing paradigm and the Grid Computing one has already been formalized in 2003, when a Global Grid Forum Research Group, called Ubicomp-RG, was established.

Some interesting work towards the realization of Pervasive Grids has been done and is reported here.

In [5], mobile devices are considered as active resources for the Grid. In particular, the authors developed a software infrastructure for deploying Grid services on mobile nodes and making them active actors in the Grid. This solution relies on a lightweight version of the .NET framework, namely the .NET Compact Framework, which enables deployment on mobile devices simple Grid Services that require limited amount of resources. It is important to note that such a solution applies only to mobile devices equipped with the Microsoft Pocket PC operating system, and requires several manual operations for installation and configuration.

In [13], the authors argue that the SOAP messaging, which is the current interaction paradigm for standard grid middleware infrastructures, is not adequate to face the needs of pervasive grids. They developed several plug-ins and a handling component for enlarging the set of available interaction mechanisms in order to make grids able to support heterogeneous software components.

Another middleware infrastructure for pervasive grids has been presented in [14]. In that instance, the authors have concentrated their effort on the extension of existing resource manager components in grid applications for making them able to register mobile devices.

2.2. Contribution

Our contribution consists of a set of basic services that realize a middleware infrastructure, namely MiPeG (*Mi*ddleware for *Pe*rvasive *G*rids), for the realization of Pervasive Grids. In particular, MiPeG extends existing Grid environments with the following characteristics:

a. *Spontaneous and transparent integration of mobile devices as active clients*. This characteristic makes the grid able to integrate mobile devices in a pervasive way; that is, without any manual configuration operation. The realized services grant access to mobile devices, which can then be used as an interfacing mechanism for the grid. In addition to this, the environment reliably handles implicit disconnections of mobile devices. Indeed, mobile users can leave the physical environment without concerning themselves about pending services. In such a case, specific services detect disconnections and handle pending computations. Some basic mechanisms for accessing the grid via mobile devices have been presented in [16].

b. *Reliable management of mobile users' sessions*. As users physically move in the environment, their running applications and their virtual environments are updated and made available for them in the new locations. This is performed by services that handle mobile sessions.

c. *Transparent integration of mobile devices as active resources*. When a mobile device enters the environment, it is automatically registered as an active resource for the grid and monitored by the environment. Consequently, the environment can allocate and execute tasks on it, as well as making services carried by the device available to grid users, in a completely transparent way for its owner. This feature, however, requires the installation of a lightweight software plug-in, which consists in an agent container, onboard the mobile device.

d. *Context-awareness*. Some mechanisms for context-awareness and location-awareness are provided. In particular, we developed services for: (i) locating mobile users and devices in a physical area; (ii) tracking their activities within the environment; and, (iii) personalizing their access based on rights, device capabilities, location, and so on.

e. *Direct and reliable execution of user code*. In classic grids users require resources; after that, they are fully in charge of launching the application, controlling it, picking up results and releasing resources. On the contrary, the Utility Computing model is based on the idea of making all execution aspects completely transparent to users; they have just to submit code and "pay" for getting results. With respect to this vision, a service for executing source Java code has been developed. In detail, users can directly submit their tasks without concerning themselves about requiring specific resources, monitoring execution, picking up results and releasing resources. All these activities are automatically performed by the service.

3. Architecture of MiPeG

MiPeG consists of a set of basic services, as shown in Fig. 1. Such services are exposed as Grid Services; i.e., they are compliant with the OGSA specifications.

It integrates with the Globus Toolkit [15], which is the de facto standard platform for Grid applications, in order to extend its functionalities and to provide mechanisms for augmenting classic grid environments with pervasive characteristics. It also partly relies on the JADE framework [22] to implement some mobile agent based components.

It is important to note that MiPeG, in addition to the classic SOAP messaging mechanism that presents several performance limitations as shown in [5], also supports an additional interaction paradigm, the publish–subscribe one, for more generic, inter-components, asynchronous communications.

MiPeG consists of the following basic services:

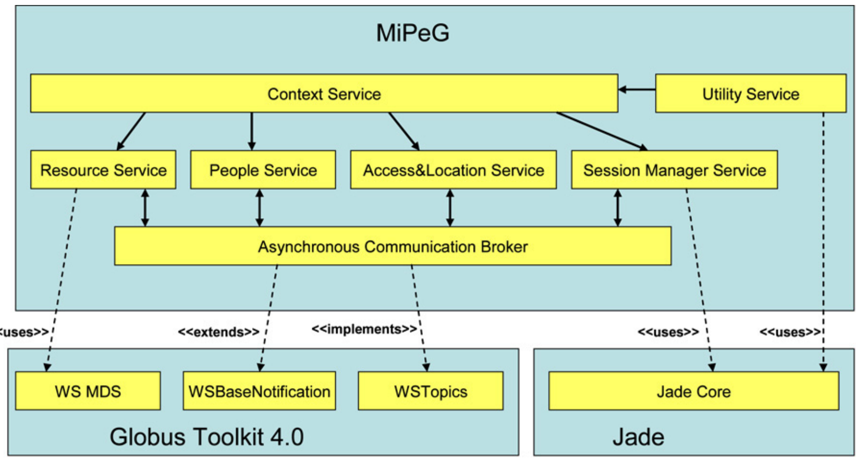


Fig. 1. Architecture of MiPeG.

• *AsynchronousCommunicationBroker*. This component is in charge of dispatching asynchronous messages in the pervasive grid. It implements the WS-BrokeredNotification specification [23]. Moreover, it extends such a specification by classifying events and handling hierarchies of classes of events. This extension provides a more flexible and efficient mechanism for subscriptions and communications.

• *Access&LocationService*. This service provides network access to mobile devices and locates them in the environment. Current implementation grants access and locates 802.11 WiFi enabled devices. It also locates RFID tagged objects. Definitively, this service is in charge of communicating to the environment (i) incoming mobile objects, (ii) location changes for active objects, and (iii) leaving objects.

• *ResourceService*. This service extends standard grid mechanisms for registering and monitoring resources, like MDS and Ganglia, in order to integrate in the grid mobile and wireless resources like laptops, PDAs and sensor networks.

• *PeopleService*. This component provides basic authentication mechanisms and handles the list of mobile users active in the environment.

• *ContextService*. The Context Service, which relies on Semantic Web technologies, handles the concept of context for a pervasive grid. Our current implementation of context consists of information related to: (i) the state of resources (both fixed and mobile) and services; (ii) the location of mobile users; and (iii) users' profiles. It is also possible to get more context information from networks of sensors that can conveniently be integrated in the environment in order to get details on ambient conditions like temperature, humidity, human presence, and so on. The service relies on ontologies and rules that enable it both to classify several typologies of entities involved in Pervasive Grids, and to infer higher-level context information from low-level information.

• *SessionManagerService*. This service handles sessions for mobile users. It consists of mobile agents that maintain the list of services activated by mobile users. A mobile agent is created when a new device comes into the environment, required to move accordingly with device movements, and destroyed when the mobile device leaves the environment.

• *UtilityService*. This service provides an implementation of the Utility Computing model. As a matter of fact, it enables users to submit source code tasks for execution. In this way, the user is freed of activities like requiring and reserving resources, launching executions, picking up results and releasing resources. It is worth noting that executions are performed directly by the environment in a transparent and reliable way.

All software elements are full open-source Java components. Moreover, most of them have a WSDL interface, consist in a GAR file generated with the Ant tool, and are deployed as Grid Services over the Globus Toolkit 4.0 platform. In addition, some services present Web interfaces, realized with the Java JSP and Java Servlet technologies, and interact with MySQL databases.

4. Details on MiPeG's services

4.1. Asynchronous Communication Broker

The *AsynchronousCommunicationBroker* provides an interaction mechanism based on the event publish–subscribe paradigm.

It implements the WSBrokeredNotification specifications. It also extends such specifications with the possibility of publishing and subscribing hierarchies of classes of events.

In particular, a class of events is a collection of events that are logically related. In such a case, rather than subscribing to every event of the class, a consumer can make just one operation of subscription for the entire class and be notified for every event.

Moreover, it handles hierarchies of classes. The concept of hierarchy has been derived from the Object Oriented inheritance technique. Since classes can be related in hierarchies, when a consumer subscribes to one class, it automatically subscribes to every upper class of the hierarchy. An example is shown in Fig. 2. In the figure, Consumer1 and Consumer2 respectively subscribe to classes C1 and C6. Because of the inheritance property of the hierarchy, Consumer1 is notified only for events appertaining to class C1. Differently, Consumer2 is notified for events of every class

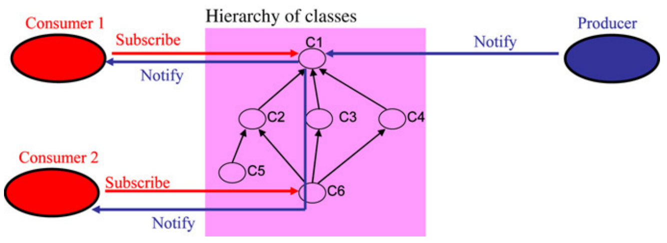


Fig. 2. A possible hierarchy of classes of events.

except C5. This is due to the fact that class C6 inherits from classes C2, C3, and C4, which in turn inherit from class C1. The figure shows the case of a producer that notifies an event of class C1. This event is dispatched both to Cosumer1 and to Consumer2.

This model represents an extension of the WSBrokeredNotification specifications. As a matter of fact, the WSBrokeredNotification specifications neither support hierarchies of classes, nor simple classes, but require that a consumer subscribe to every specific event for which it is willing of being notified.

In Fig. 3, the WSDL interface of the *AsynchronousCommunicationBroker* is shown. The Broker exposes functions for subscribing and unsubscribing classes of events, notifying events, creating and handling hierarchies. In addition, two hierarchies, which will be used in the running scenario, are detailed. In particular, hierarchy 1 consists of three classes. Class MOBILITY contains the event NEW_LOCATION. Class PRESENCE contains the events NEW_DEVICE and DEVICE_HAS_LEFT. Finally, class LOCALIZATION groups all events. Hierarchy 2 consists of a single class, namely USER_PRESENCE, which contains events for logging in and out a user.

4.2. People Service

This service provides basic authentication mechanisms and handles active users in the environment. As shown in Fig. 4, it exposes a Web interface for authenticating connected users and for handling authorized users. An authorized user can log in by means of a JSP application form, and log out either explicitly or implicitly by closing the web page. Moreover, some administrative functions are available. The environment can interact with the service via the WSDL interface in order to get the list of active users or the rights of a specific user.

The service also notifies to the environment events of the class USER_PRESENCE whenever a user logs in or out.

4.3. Access&Location Service

This service is able to locate active mobile objects like WiFi enabled devices and RFID tagged entities [17]. It offers both locating and location functions; that is, the function *Locate_Object* returns the position of a specific object, whereas the function *Get_Objects* returns the list of objects that are active at a specific location (see Fig. 5).

It notifies to the environment events of the class LOCALIZATION.

In addition to location and locating functions, this service provides basic network facilities for connecting incoming mobile devices and detects leaving mobile objects.

The service architecture consists of two layers and the following components:

• *WiFiLocatingComponent*. This component is in charge of locating WiFi enabled mobile devices. In particular, a WiFi location is identified by the area covered by a specific wireless Access Point (AP). In the environment, one *WiFiLocatingComponent* is deployed per every wireless AP. Our current implementation uses 3Com Office Connect Wireless 11g Access Points. Whenever a mobile device connects with the AP, the AP writes an event in its log file. The *WiFiLocatingComponent* periodically interrogates such a log file and communicates them to the *LocationComponent* when a new device has connected. The *WiFiLocatingComponent* maintains information on devices locally connected to its AP.

• *RFIDLocatingComponent*. This component is in charge of locating RFID tagged objects. An RFID location is identified by the area covered by a specific RFID reader. Current implementation uses the passive, short-range (30 cm), Feig Electronic RFID, model ISC.MR 100/101. When a tagged object enters the area covered by an antenna, the RFID reader generates an event that is caught by the *RFIDLocatingComponent*. Then, the *RFIDLocatingComponent* communicates to the LocationComponent such an event.

• *DHCPComponent*. This component implements a DHCP service. It provides network connectivity to incoming IP enabled devices as a standard DHCP, but it has also additional functionalities. In particular, it is able to release an IP address on demand. In this way, if a device has left the environment, the *LocationComponent* requires that the *DHCPComponent* free the IP address of that device.

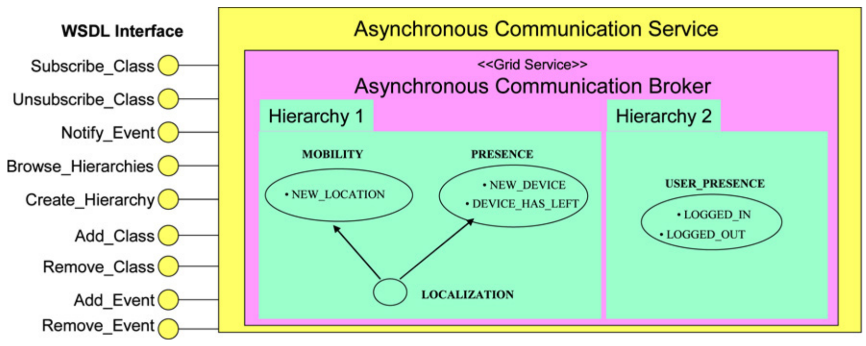


Fig. 3. Interfaces and architecture of the ACB.

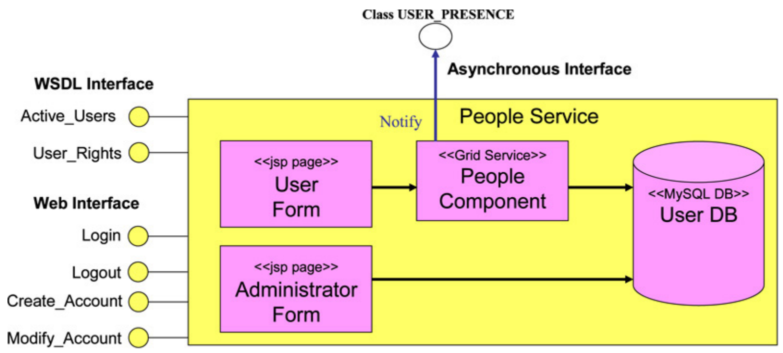


Fig. 4. Interfaces and architecture of the *PeopleService*.

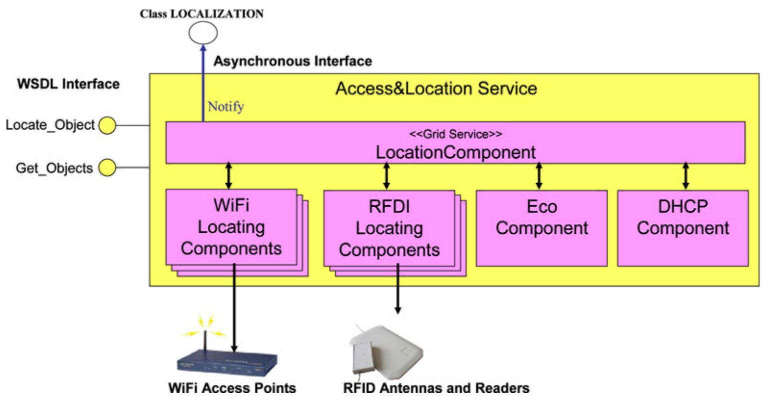


Fig. 5. Interfaces and architecture of the *Access&LocationService*.

• *EcoComponent*. This component sends ping messages towards mobile IP devices in order to detect leaving objects. When an implicit disconnection is detected, the component communicates such an event to the *LocationComponent* that notifies the DEVICE_HAS_LEFT event.
• *LocationComponent*. This component is in charge of handling global location states obtained by combining information coming from Locating components. When a mobile object changes its position, a NEW_LOCATION event is dispatched. Moreover, when a mobile object is detected for the first time, the *LocatingComponent* notifies such an event (NEW_DEVICE) to the *ResourceService* that updates its registers.

4.4. Context Service

The *ContextService* exploits Semantic Web technologies to support the explicit representation of context and reasoning in the Pervasive Grid [18].

In this scenario, context groups' information related to concepts like people, devices, resources, services, locations, physical environment conditions, etc.

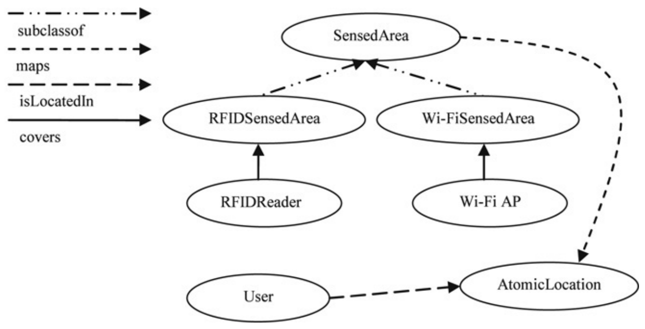
It is also important to note that the *ContextService* handles several semantic aspects. As an example, consider the problem of determining the position of mobile devices, as well as the position of RFID tagged objects, within the environment. Positioning systems return information like "*The mobile device Alpha is connected to the Access Point X*" or "*The tag Beta has been detected by the RFID reader Y*"; whereas, high level application services need responses to questions like "*Which devices are active in the multimedia laboratory?*" or "*Is the mobile user John in the meeting room?*". From this point of view, the necessity of having a service able to relate information on physical locations appears clear(especially Access Point X, RFID Reader Y, etc.) to semantic locations (Multimedia Laboratory, Meeting Room, etc.).

On the other hand, the *ContextService* integrates different positioning systems and infers higher level semantic information from events generated by positioning systems. For example, when an RFID-tagged user approaches an RFID reader carrying with him his mobile device, the *ContextService* is able to determine, by performing some reasoning, that also the mobile device is located in a more restricted area (the one covered by the RFID reader) instead of a larger location (the one covered by an Access Point). As a result, a better resolution for the localization of the mobile device is obtained.

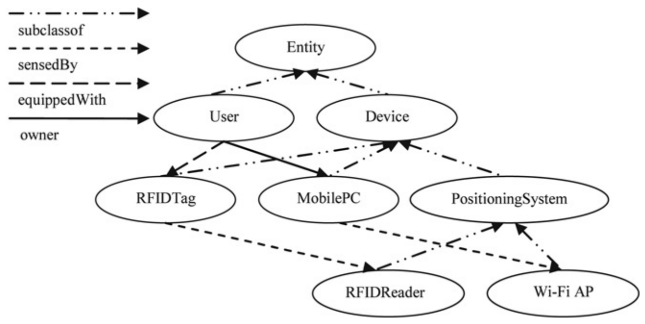
Fig. 6 shows the ontology that defines the context and some of the rules that the *ContextService* applies to infer higher semantic information.

The main component of the *ContextService* is the *ContextReasoningEngine*, which is in charge of (i) handling the context, (ii) gathering information coming from *SensingComponents* and other services; and (iii) making inferences based on context information.

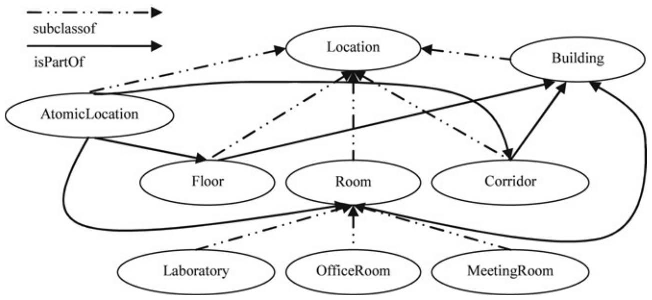
The architecture of the ContexService is described in Fig. 7. It consists of a two layers of components and a *ContextModel*. The *ContextModel* is defined by the ontology and the rules reported in the previous figure. The *SensingComponents* are



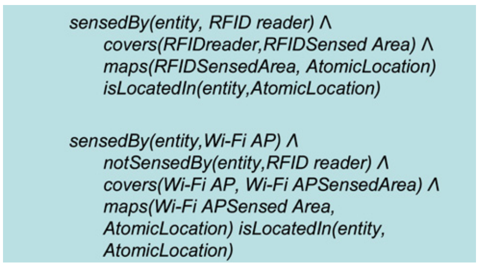
(a) Ontology for physical areas.



(b) Ontology for entities.



(c) Ontology for semantic locations.


sensedBy(entity, RFID reader) ∧
covers(RFIDreader,RFIDSensed Area) ∧
maps(RFIDSensedArea, AtomicLocation)
isLocatedIn(entity,AtomicLocation)

sensedBy(entity,Wi-Fi AP) ∧
notSensedBy(entity,RFID reader) ∧
covers(Wi-Fi AP, Wi-Fi APSensedArea) ∧
maps(Wi-Fi APSensed Area,
AtomicLocation) isLocatedIn(entity,
AtomicLocation)

(d) Some semantic rules.

Fig. 6. Ontologies and semantic rules.

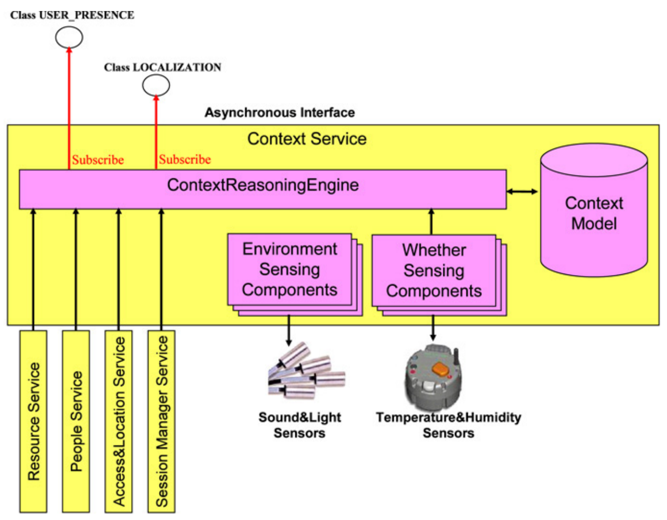


Fig. 7. Interfaces and Architecture of the *ContextService*.

the ones in charge of driving the sensor networks. by contrast, the *ContextReasoningEngine* is the component that applies reasoning mechanisms on the model.

To summarize, the key features of the *ContextService* are:

- Semantic integration of different sensors and positioning systems;
- Definition of a context model;
- Logic reasoning on context information to get higher-level semantics.

4.5. Resource Service

The *ResourceService* extends standard grid mechanisms – specifically the MDS-register service [15] and the glue schema [21] – for registering and monitoring the grid's resources. In particular, extensions enable the grid to handle a large set of mobile devices, which can vary from laptops to PDAs and sensor networks.

As a matter of fact, mobile devices and sensor networks have specific features, which distinguish them from classic fixed grid resources, not taken into account by classic tools. As an example, in order to monitor the state of a mobile device, it is essential to know the type of alimentation: battery or fixed; in the former case, the level of the battery; the type of network connection (a lot of new kinds of wireless connections like WiFi, Bluetooth, GPRS, UMTS, etc.); the type of operating system (new operating systems like Symbian, Pal OS, MS Pocket PC, Pocket Linux, etc.).

New elements are reported in Fig. 8, which shows the information model obtained by extending the standard glue schema with new classes.

Our service architecture consists of the following components (see Fig. 9):

- *ResourceComponent*. This component handles the extended version of the glue schema and the list of available resources, which are registered in the GT4 MDS-Index Service. In addition, the *ResourceComponent* receives operating information by information providers and updates the resources' states.
- *LaptopInfoProvider*. This component is in charge of monitoring the state of mobile laptops. It is a specialization of the Ganglia tool [20] and handles parameters like type of alimentation, battery level, location, operating system, network interface, etc.

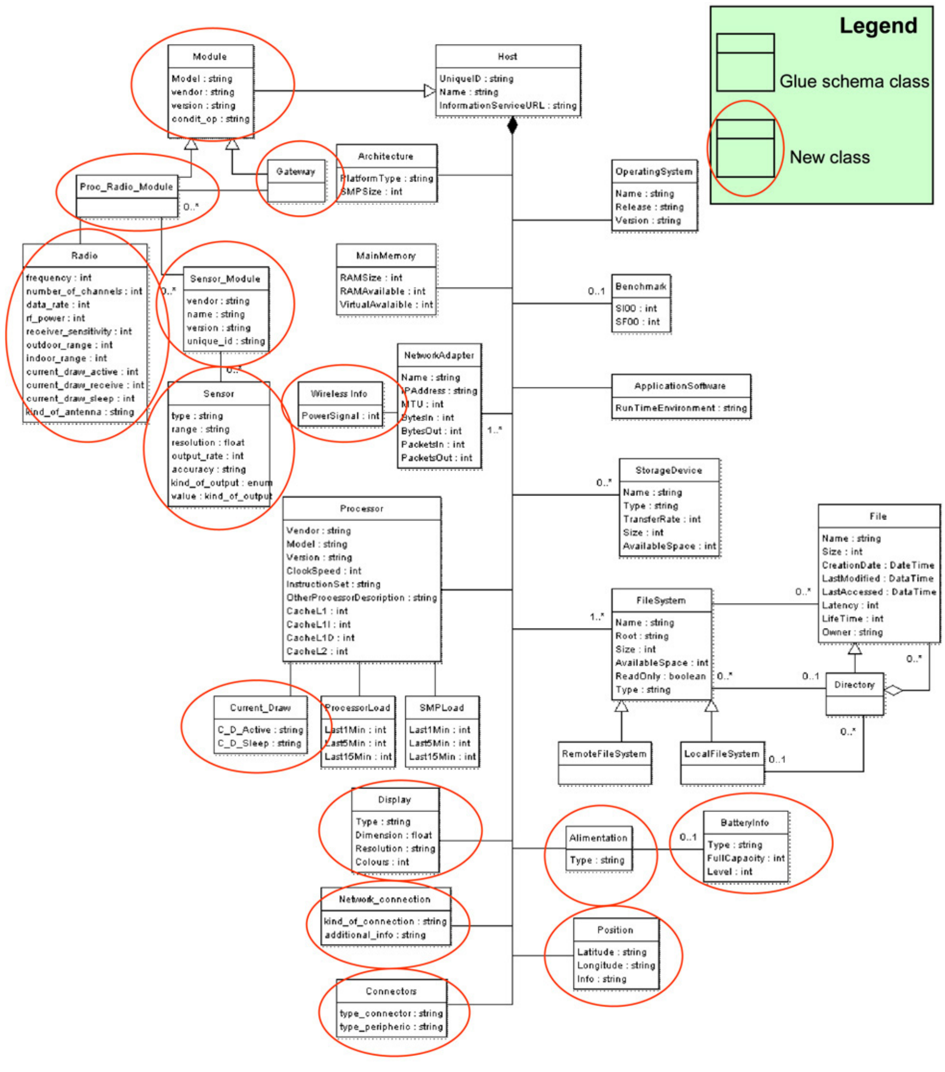


Fig. 8. Information model.

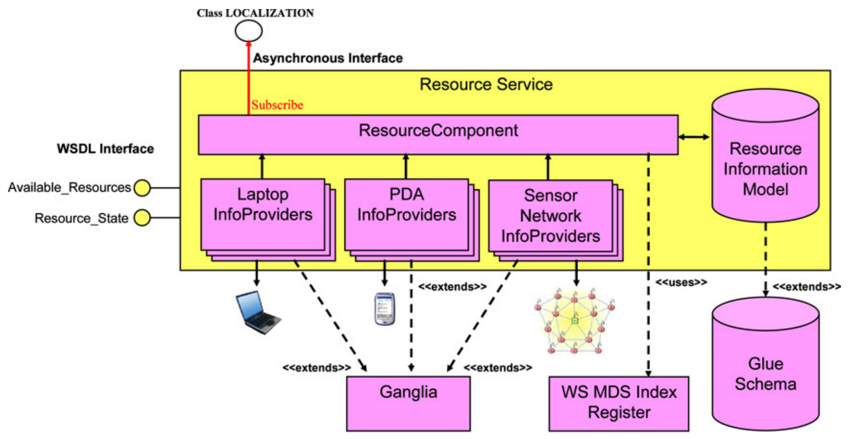


Fig. 9. Interfaces and architecture of the *ResourceService*.

• *PDAInfoProvider*. This software element is also an extension of the Ganglia tool that monitors the state of mobile PDA.
• *SensorNetworkInformationProvider*. This component is in charge of monitoring the state of mobile laptops. Again it has been developed as an extension of the Ganglia tool that handles the parameters obtained by a sensor network like the number of nodes, temperature, humidity, etc.

It is important to note that, in accordance with the pervasive model, an incoming mobile device is transparently registered as

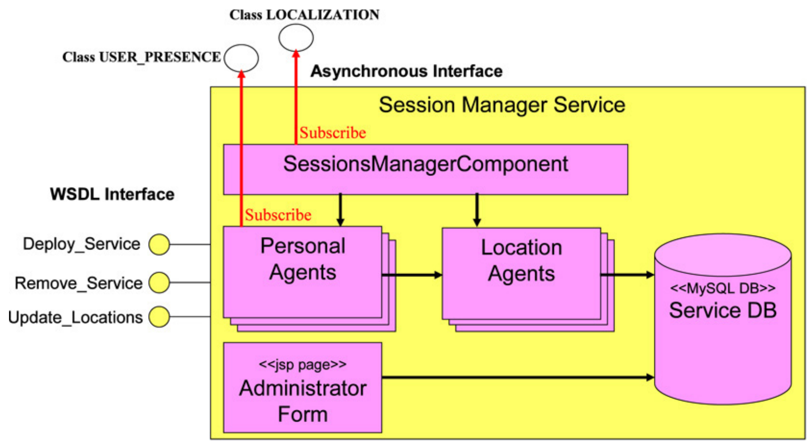


Fig. 10. Interfaces and architecture of the *SessionManagerService*.

an available resource as soon as it enters the environment and is detected by the *Access&LocationService*.

4.6. Session Manager Service

The *SessionManagerService* has two major functions. It handles both the list of services available at every location of the environment (in a pervasive scenario some services could not be available everywhere, but only at specific locations) and the list of services activated by every user.

As shown in Fig. 10, it consists of the following components:

• *LocationAgent*. This is an agent deployed at a specific location. There's a *LocationAgent* at every location. It handles the list of application services and resources available at its location. Service availability may depend not only on the physical location but also on user's rights and profile. In particular, several levels of access are defined.

• *PersonalAgent*. This is a mobile agent created when a new device appears in the environment. In particular, the *SessionManagerComponent*, which is notified for the NEW_DEVICE event, creates the *PersonalAgent* at the location where the mobile device has been detected. After its creation, the *PersonalAgent* subscribes to the class USER_PRESENCE and interacts with the local *LocationAgent* to have the list of available services. From this moment on, the *PersonalAgent* catches a user's requests and maintains the list of services activated by him. In the case the user authenticates himself, the *PersonalAgent* is notified by the LOGGED_IN event, then interacts once again with the *LocationAgent* to produce the new list of available services. When a device moves to a new location, the *SessionManagerComponent*, which receives the NEW_LOCATION event, requires that the *PersonalAgent* migrate and update the list of services available at the new location. Finally, in case the user logs out or his device disconnects, the *PersonalAgent* handles any pending computations.

• *SessionManagerComponent*. This component is an agent container. It creates both Personal and Location Agents. It also drives Personal Agents to move in tune with their owners' movements. In particular, it creates a new *PersonalAgent* when it receives a NEW_DEVICE event; it requires that the *PersonalAgent* move to a new location when it catches a NEW_LOCATION event; finally, it requires that the *PersonalAgent* destroy itself when the DEVICE_HAS_LEFT event is notified.

All components for the *SessionManagerService* have been developed over the JADE framework, which is fully compliant with the FIPA specifications [19].

Finally, it is worth noting that the latency (time between the moment when the location service detects the mobile user/device in the new location and the moment when the *PersonalAgent* is presented to the user) is less then 2 s both for the case of RFID localization and the case of WiFi localization.

4.7. Utility Service

The *UtilityService* is an application service able to dynamically distribute and execute a user's tasks on a grid of both fixed and mobile resources. In line with the Utility Computing model, users willing of executing their applications directly submit their code without caring of choosing and allocating resources of the grid.

It is worth noting that the *UtilityService* is not a workflow system like Pegasus [19] or K-WF Grid [25], which enable one to compose Grid services by specifying a workflow. Differently, the *UtilityService* provides mechanisms for submitting source code without the need/possibility of specifying workflows. This is due to the fact that it has been devised to support users and other services that require the execution of source code; for example, the case of a software testing service (described in the experimental scenario), which provides a web interface for submitting the source code to test and the set of test cases.

After having been submitted, tasks are completely handled by the environment, which gathers the results and sends them back to the user.

To achieve this objective, user tasks are encapsulated in mobile agents and then allocated in a distributed platform that controls execution.

As shown in Fig. 11, the *UtilityService* consists of the following components:

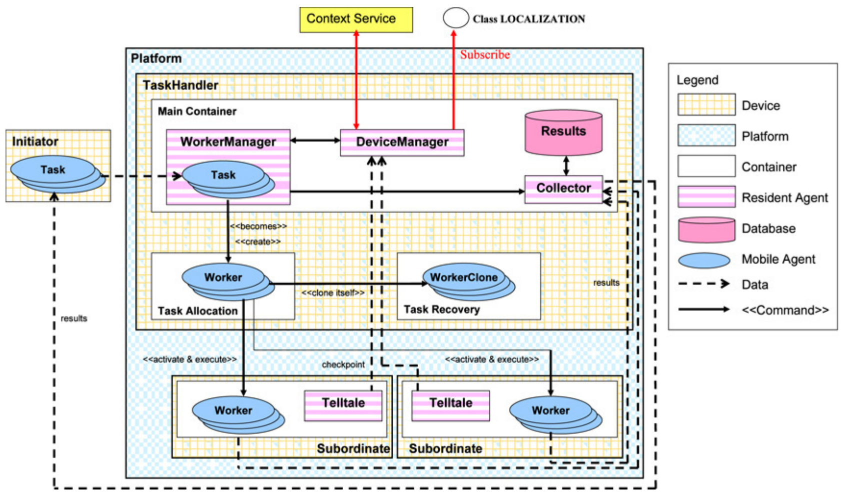


Fig. 11. Interfaces and Architecture of the *UtilityService*.

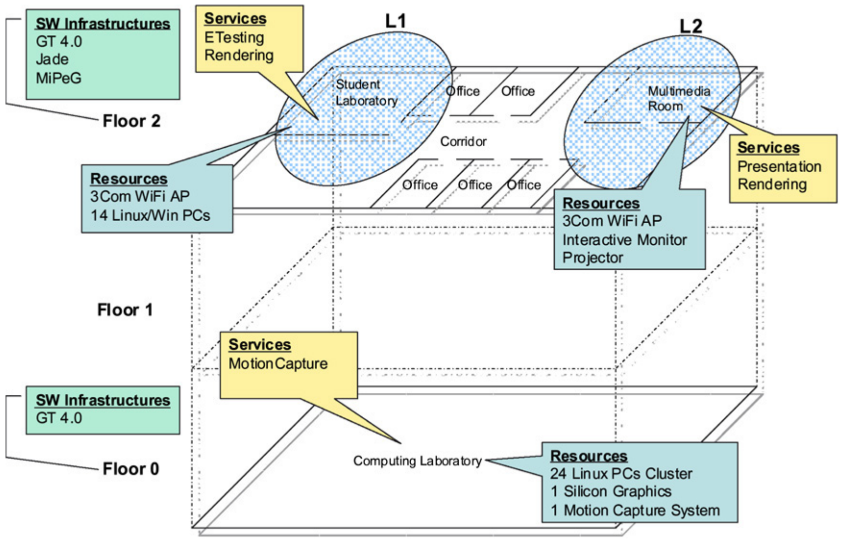


Fig. 12. Real environment.

• *Platform*. This is the set of hardware nodes equipped with a *Container* and able to execute tasks;

• *TaskHandler*. This is the hardware element that hosts the coordinating components. It is also the user's entry point to the service;

• *Subordinate*. This is a hardware node that hosts mobile agents for execution. It can be either a fixed or a mobile resource;

• *Initiator*. This is the hardware element used by the user to submit source code for execution;

• *Container*. This is the run-time software environment that provides the basic functionalities for executing and coordinating mobile agents;

• *TaskAllocation*. This is the *Container* that handles the mobile agents hosting user tasks;

• *TaskRecovery*. This is the *Container* that stores cloned mobile agents. It is required that clones be activated in case of any failure of the cloned agent;

• *Worker*. This agent encapsulates the user task for execution and sends execution results to the *Collector*. More *Workers* can be hosted by the same *Subordinate*;

• *DeviceManager*. This agent interacts with the *ContexService* to receive the list and the state of available resources in the grid. In addition to this, it receives checkpoints from every *Subordinate*.

• *Telltale*. This is a software element that monitors some of *Subordinate*'s parameters and communicates them to the *DeviceManager*;

• *WorkerManager*. This agent coordinates *Workers* allocation and migration within the environment in accordance with a scheduling algorithm;

• *Collector*. This agent receives the results from every active *Worker* and collects them in the *Results* archive;

• *Results*. This is the archive that stores execution results until they are sent back to the user.

Whenever a user wants execute a task, she contacts the *Initiator* and submits her code. After that, the *Initiator* embeds such a code in a mobile agent, namely a *Worker*, into the *TaskAllocation* container ready to be executed. Before distribution and execution, the task is forced to clone itself and the clone is inserted in the *TaskRecovery* container. This is performed in order to confer a certain degree of dependability to the service. Next, the task is allocated in one or more *Subordinates*, which will execute them and produce results. Allocation is driven by the *DeviceManager* depending on the current state of active resources of the grid. From now on, two main possibilities are in order. The *Worker* completes its execution by sending results to the *Collector* which, in turn, stores them in the *Results* archive; or, the *Worker* fails. In the latter case, failure is detected by the *DeviceManager* that doesn't receive checkpoints from the *Telltale* anymore. As a consequence, the *DeviceManager* activates the clone and requires its execution on a new *Subordinate*.

5. Experimental scenario

The experimental scenario consists of a physical site located in a three-floor building. The virtual environment uses two floors of the building (see Fig. 12).

Floor zero has a computing laboratory in which a cluster of 24 Linux PCs, a 12-processor Silicon Graphics workstation, and a motion capture system are deployed. These resources are collected in a wired grid built on top of the Globus Toolkit 4.0 platform.

On floor two, wireless access to the grid is available. As a matter of fact, two 3Com Office Connect Wireless 11g Access Points identify two distinct locations. L1 is a student laboratory where our students develop their activities and periodically perform E-Tests. L2 is a multimedia room equipped with a projector, an interactive monitor, and other multimedia devices.

Some application services are available:

• *MotionCaptureService*. This service relies on the motion capture system. An actor (equipped with optical markers) moves around in the multimedia laboratory. Several cameras capture his movements, which are reproduced on a graphic station. The graphic station shows a skeleton, which moves in step with the actor, and records data movements in a file;

• *RenderingService*. This service enables users to submit row motion data and to build 3D graphic applications. This service is exposed as a Grid Service and is available at every location (L1, L2);

• *PresentationService*. This service enables a user to project her presentation in the multimedia room. The service receives a pdf/ppt file via a dialog form and then enables speaker to control the presentation flow. This is an interactive service, which requires the speaker to be in the room for the presentation. As a consequence, the service must be available only in the multimedia room (L2);

• *ETestingService*. This service performs on-line evaluation tests for courseware activities. When a session test starts, students must be in the student laboratory. Evaluation tests are synchronized and students have a predefined period of time for completing each test section. Students can interrupt their test by explicitly closing the service or by leaving the multimedia room. This service is exposed as a Grid Service, but it must be available only in the student laboratory (L1);

• *SoftwareTestingService*. This service, which relies on the *UtilityService*, performs source code software testing. In particular, it offers a web interface for submitting source code and test cases. After having received the source code and test cases from the *SoftwareTestingService*, the *UtilityService* generates a task for every test case. Tasks are then embedded within Worker Agents and deployed on a device for execution. In this way, the *SoftwareTestingService* enables software developers to conduct parallel software tests of the same branch of code, even for hundreds test cases.

The environment was formerly created with classic Grid infrastructure deployed over the Globus Tolkit platform. Successively, it has been enhanced with characteristics coming from the pervasive computing paradigms, and with new services that support mobile technologies. In the current implementation of the environment, service availability depends on a user's location, rights, and context. As a matter of fact, we can report some example scenarios:

1. The *PresentationService* is available only to users that are located in L2. In particular, a mobile user, who moves in location L2, is followed by his *PersonalAgent*. The *PersonalAgent* interacts with the *LocationAgent* and updates the list of available services. From now on, the mobile user can get access to the *PresentationService*.

2. The *ETestingService* is available for every authenticated mobile user within the student laboratory. In particular, while a test session is active, a student has two possibilities in order to get access to the service: she has an RFID tag and approaches a wired station (all wired stations in the lab have an RFID reader), so that she is identified and located by the *RFIDLocationComponents*; or, the student enters the lab with her own mobile device, then she is located by the *WiFiLocatingComponent* and gets access after having authenticated herself. On the other hand, if the student leaves her wired station (in the case of RFID tagged users) or the location L1 (in the case of mobile students with their own device), the *ETestingService*, which is notified by the *Access&LocationService*, automatically disconnects the leaving user and denies any further attempts at reconnection.

3. The *RenderingService* is available for authorized users at every location. However, the QoS is improved by context-awareness. Indeed, in the case a mobile user launches such a service while in location L1, the rendered data are reproduced on his mobile device. After that, if the user moves in the multimedia room (L2), any rendered data are automatically switched on the interactive monitor (if idle).

We have monitored the access to application services for three weeks. As reported in Fig. 13, we have registered 155 total accesses and 49 accesses through mobile devices (32%). Fig. 13 reports our results. It is worth noting that access through mobile devices is granted because the application services are equipped with multi-channel interfaces and because MiPeG enables both mobile devices to connect in a transparent and spontaneous

Service	Mobile Accesses	Wired Accesses	Total Accesses
Motion Capture Service	0	4	4
Rendering Service	7	20	27
Presentation Service	6	0	6
E-Testing Service	36	82	118
Total	49	106	155
Percentage	31,61%	68,39%	

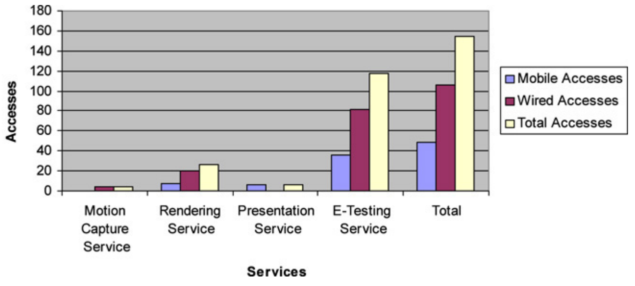


Fig. 13. Number of accesses to grid services.

way, and the environment to efficiently handle and locate them. Therefore, a classic grid infrastructure wouldn't be able to offer services to mobile devices.

In the previous examples, we have shown some results on the accessibility of grid services by mobile users.

With more recent experiments, we have concentrated on the reliability aspects. In this case, preliminary results have shown that the percentage of mobile sessions lost with respect to the changes of location for the mobile users is less then 14%.

Finally, some functional verifications for the *SoftwareTestingService* have been performed by soliciting the service with software elements and hundreds of test cases. For these experiments, the *SoftwareTestingService* has been configured to use only mobile devices in order to obtain useful computing power from resources typically neglected by classic grids. Experiments have shown that the service is able to achieve its objective and return results with a good response time. However, a deeper experimental analysis must be conducted in order to get performance measures for a wider number of tasks submitted.

6. Conclusions

This paper has described a middleware for pervasive grids. The middleware provides a set of basic services for: (i) integrating mobile devices in the grid; (ii) for extending the grid with context and location-awareness; and, (iii) providing an implementation of the Utility Paradigm.

Integration of mobile devices has taken place both for enabling mobile users to get access to grid services in a pervasive way, and for extending the set of available grid resources to mobile equipments and sensor networks.

References

[1] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.

[2] D. Saha, A. Murkrjee, Pervasive computing: A paradigm for the 21st century, IEEE Computer (March) (2003).

[3] A. Litke, D. Skoutas, K. Tserpes, T. Varvarigou, Efficient task replication and management for adaptive fault tolerance in mobile grid environments, Future Generation Computing Systems 23 (2) (2007) 163–178.

[4] L.W. McKnight, J. Howinson, S. Bradner, Wireless grids, IEEE Internet Computing (July–August) (2004).

[5] S. Oh, G.C. Fox, Optimizing Web Service messaging performance in mobile computing, Future Generation Computing Systems 23 (4) (2007) 623–632.

[6] F.J. González-Castaño, J. Vales-Alonso, M. Livny, E. Costa-Montenegro, L. Anido-Rifón, Condor grid computing from mobile handheld devices, ACM SIGMOBILE Mobile Computing and Communications Review Archive 7 (1) (2003).

[7] Utility computing, IBM Systems Journal 43 (1) (2004).

[8] D.C. Chu, M. Humphrey, Mobile OGSI.NET: Grid computing on mobile devices, in: International Workshop on Grid Computing, GRID, 2004.

[9] B. Clarke, M. Humphrey, Beyond the 'device as portal': Meeting the requirements of wireless and mobile devices in the legion of grid computing system, in: International Parallel and Distributed Processing Symposium, IPDPS, 2002.

[10] T. Phan, L. Huang, C. Dulan, Challenge: Integrating mobile devices into the computational grid, in: International Conference on Mobile Computing and Networking, MobiCom, 2002.

[11] N. Daves, A. Friday, O. Storz, Exploring the grid's potential for ubiquitous computing, IEEE Pervasive Computing (April–June) (2004).

[12] V. Hingne, A. Joshi, T. Finin, H. Kargupta, E. Houstis, Towards a pervasive grid, in: International Parallel and Distributed Processing Symposium, IPDPS, 2003.

[13] G. Coulson, P. Grace, G. Blair, D. Duce, C. Cooper, M. Sagar, A middleware approach for pervasive grid environments, in: Workshop on Ubiquitous Computing and e-Research National eScience Centre, Edinburgh, UK, 18–19 May, 2005.

[14] C.F.R. Geyer, et al., GRADEp: Towards pervasive grid executions, in: III Workshop on Computational Grids and Applications, WCGA, 2005.

[15] I. Foster, Globus toolkit version 4: Software for service-oriented systems, in: IFIP International Conference on Network and Parallel Computing, in: LNCS, vol. 3779, Springer-Verlag, 2005, pp. 2–13. Also available online at: www.globus.org.

[16] A. Coronato, G. De Pietro, M. Ciampi, Middleware services for pervasive grids, in: Proc. of the 4th International Symposium on Parallel and Distributed Processing and Applications, ISPA06, in: Lecture Note in Computer Science, LNCS, vol. 4330, Springer Verlag.

[17] A. Coronato, G. Della Vecchia, G. De Pietro, An RFID-based Access&Location Service for pervasive grids, in: Proc. of the 1st International Workshop on Thrustworthiness, Reliability and Services in Ubiquitous and Sensor neTworks, TRUST 2006, in: Lecture Notes in Computer Science, LNCS, vol. 4097, Springer Verlag.

[18] A. Coronato, G. De Pietro, M. Esposito, A semantic context service for pervasive grids, in: Proc. of the International Conference on Hybrid Information Technology, ICHIT 2006, IEEE CS.

[19] G. Singh, et al., The pegasus portal: Web based grid computing, in: Proc. of the 2005 ACM symposium on Applied computing, SAC2005, ACM.

[20] http://ganglia.sourceforge.net/.

[21] http://glueschema.forge.cnaf.infn.it/.

[22] F. Bellifemmine, A. Poggi, G. Rimassa, Jade programmers guide. http://sharon.cselt.it/projects/jade.

[23] http://www.oasis-open.org/committees/download.php/13485/wsn-ws-brokered_notification-1.3-spec-pr-01.pdf.

[24] http://www.cs.wisc.edu/condor/.

[25] http://www.kwfgrid.net/.

[26] I. Foster, C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publisher.

Antonio Coronato is Researcher at the Institute for the Development and Application of Territorial Information Systems (SASIT) of the National Research Council (CNR). He is a contract professor of Software Engineering at the University of Naples "Federico II". His main fields of interest are related to pervasive computing and component based architectures. He is member of the ACM.

Giuseppe De Pietro is a Senior Researcher at the Institute of High Performance Computing and Networking (ICAR) of the National Research Council (CNR). He is a contract professor of Information Systems at the Second University of Naples. His research interests cover pervasive computing, multimodal and virtual reality environments. He is member of the IEEE.