J Internet Serv Appl (2011) 2:155–170
DOI 10.1007/s13174-011-0030-2

SI: CLOUD COMPUTING

MITSIO—an architecture for the management of interactive tasks and the semantic integration of ontologies in the mobile grid

Vinicius C.M. Borges · A.G.M. Rossetto · A.P.C. Silva · M.A.R. Dantas

Received: 8 November 2010 / Accepted: 27 June 2011 / Published online: 5 August 2011

Abstract The interaction between mobile and grid environments in building a transparent infrastructure for mobile users, involves several factors from both wired and wireless networks. These include the following: coordinated application executions, the management of inherent characteristics of mobile devices (disconnections and the problem of a reduced battery lifetime), and making a selection of suitable heterogeneous resources in wired network, which are distinctly in terms of their virtual organizations. For this reason, the main goal of the architecture is the provision of more complete transparency for mobile users, when they need to access the processing capacity of the grid computing environment. To achieve this, this paper outlines an architecture, called MITSIO, together with a prototype implementation that takes these factors into consideration. Furthermore, the study examines two case studies that provide evidence of the capacities of the proposed architecture. The first case study shows how the architecture enables mobile devices to consume less battery power (approximately 31%, for submitting and monitoring applications). The second case provides evidence of its capacity to select resources from different virtual organizations through the semantic integration of multiple ontologies.

Keywords Mobile grid · Disconnection · Application workflow · Transparency · Semantic integration

V.C.M. Borges (✉) · A.G.M. Rossetto · A.P.C. Silva · M.A.R. Dantas
Laboratory of Research in Distributed Systems (LaPeSD), Department of Informatics and Statistics (INE), Federal University of Santa Catarina (UFSC), 88040-900 Florianopolis, SC, Brazil
e-mail: vcunha@inf.ufsc.br

A.G.M. Rossetto
e-mail: anubis@inf.ufsc.br

A.P.C. Silva
e-mail: parra@inf.ufsc.br

M.A.R. Dantas
e-mail: mario@inf.ufsc.br

1 Introduction

The resolution of complex problems is becoming possible as a result of the continuous evolution that is being observed in hardware and software. Thus, a parallel computing paradigm has been employed in this study and this involves heterogeneous high performance resources, such as those found in grid or cloud computing. It is worth noting that cloud computing shares certain features with mobile grid computing, such as distributed resources to achieve application-level objectives and ubiquitous access for end-users. The limitations that are usually found in mobile devices make it very difficult to provide mobile users with the means of solving complex problems when using these devices. As a result, it is of great value to find a way to integrate mobile devices with grid environments, and thus build an infrastructure for cloud computing [2, 26, 40].

According to [2, 8, 15, 26], mobile grid environments can have two kinds of interaction: the devices can be regarded as users of grid resources (i.e., grid interfaces) or as the grid resources themselves (e.g., processor power, disk space, cameras, microphones, GPS receivers, accelerometers, and sensors). On the one hand, the computing power of the mobile devices has witnessed an increase in the rate of improvement in recent years. However, despite this, the current capacity for processing and storage has not been enough to allow an adequate resolution of the complex problems within these devices. On the other hand, grid computing and cloud

computing are easily accessible through a variety of mobile devices with Internet connections. It should also be noted that the mobile devices play a key role in improving ubiquitous access to the grid and cloud configurations. For these reasons, the architecture outlined in this article extends the computing power of the grid environment to mobile users and in this research study, mobile devices can be regarded as web interfaces on condition that the grid resources exist in wired networks.

In order to enable a better integration between the mobile and grid environments, any proposal should consider the specific challenges from the mobile grid environment such as management of mobile devices, transparency, security, and context-awareness. As [11, 28, 40] observes, it is important to note that fixed or mobile interfaces can conceal the complexity and technical aspects of the mechanisms and principles of each mobile device, grid, and cloud environment, such as tasks submission and resource selection. Hence, to provide more complete transparency for the mobile users, when they access these distributed and complex configurations, the approaches should take into consideration both wired and wireless networks factors. There is a lack of approaches which provide a more complete transparency for mobile users, and thus to offer ubiquitous grid access with greater transparency is still an open issue. Therefore, the challenge of transparency is the main issue that will be addressed in this paper, since it is an important requirement when mobile devices are used as an interface to access the grid. For this purpose, an architecture is outlined, called *MITSIO (**M**anagement of **I**nteractive **T**asks and **S**emantics **I**ntegration of **O**ntologies)*; this handles three factors of transparency, which need to be examined to improve the provision of transparency for the mobile user. The aspects of transparency that are dealt within this paper are as follows: management of interactive tasks (submission and monitoring of workflows), resource selection in the wired network and the management of inherent characteristics of mobile devices (disconnections and the problem of a reduced battery lifetime).

The article is organized as follows: some relevant questions arising from related research studies are discussed in Sect. 2. Section 3 deals with factors concerning the architecture, such as the design, implementation of the prototype and two case studies. The first case study (Sect. 3.3.1) draws a comparison between the battery power consumption of the PDAs and the MITSIO architecture proposed for the application submissions; this is in contrast with the approach adopted for submitting applications in many related works. Following this, there is a case study (Sect. 3.4.4) which describes the capacity of the Resource Selector component to select the heterogeneous resources, which are described distinctly by their virtual organizations. Finally, in Sect. 4, some conclusions are discussed and recommendations made for further research.

2 Related works

The literature shows that related research works (examples can be found in [5, 6, 12, 16, 17, 31, 33], usually fail to take account of how the submission and monitoring of applications can be used to solve an isolated problem in an automated and coordinated manner through the workflow concept, by making use of the available grid resources. These research works allow a task submission and monitor to occur at the same time in the interface of the device. Consequently, the mobile user has to control the submission order of several tasks.

Owing to the mature state of the grid computing paradigm, which provides many kinds of resources and, therefore, the capacity to solve complex problems, it has become more common for the application execution to carry out multiple tasks for the resolution of each problem. In general, these applications have (loop) iterations and dependencies (i.e., they require user interactions), which rely on the use of certain computational tools (e.g., software and/or database) that are shared by virtual organizations in a grid configuration or even the infrastructure providers in cloud environments. These tasks represent a workflow the data of which are sent/received when they are carried out and comply with certain rules. Thus, there is a need to employ a mechanism to control, organize, and allow a degree of automatization in the execution of the task-flow. For this reason, the workflow concept is employed in our proposal, in a similar way to some other research projects [20, 32, 34]. This concept offers thin granularity and a generic solution for the definition of the resources used in each stage of the application execution in an automated and coordinated behavior. The classic concept of workflow is shown in [13].

On the other hand, the approaches that are outlined in [1, 14], suggest that the PDA client can submit several tasks that can be combined to solve a problem, by employing the workflow concept. However, these approaches show clearly that PDA clients implement a small set of workflow functionalities, although they do not show functionalities that are implemented. For example, no mention is made of any aspect of transparency for the mobile users. The main focus of [1] is on the security implementation in the PDA clients for tasks submission in UNICORE middleware, where a safe communication between clients and grid middleware is mandatory. In [14], the main focus is on the discovery of workflows in data mining grids for submission from the PDA. In [16, 17], the authors pursue proxy approaches based on a coordinated checkpointing scheme for fault recovery. However, these proxy approaches do not fulfill the task of submitting application workflows and as a result, do not handle the disconnections that are needed to adapt the execution flow to ensure the application is carried out in a consistent way.

The research study shown in [12] tackles the problem of giving mobile users the chance to access interoperable services based on ontology (e.g., resource management, authorization service, and information service), since these mobile devices are subject to resource restrictions. Hence, it provides these services in the mobile devices through simple interfaces. However, this approach does not take into account that the resource selection can be found in different virtual organizations which have distinct ontologies to describe the grid resources. Furthermore, it also fails to allow the submission and monitoring of application workflow or the disconnection of mobile devices, while executing the submitted application.

In addition, in [6, 33], there are proposed architectures that provide methods for dealing with interactions between the pervasive devices and a grid in order to exploit the grid resources in the wired networks, and thus carry out resource-intensive tasks. These architectures provide a lightweight portal for the mobile devices, where the ontologies can assist the users to discover services and resources and also enable the submission of applications workflows to occur. However, they fail to take account of the disconnections of the mobile devices that may occur during the execution of the workflow. Furthermore, the authors in [33] point out that there is a need for an efficient resource selector mechanism to match the workflow application semantically, when the workflow application submitted by the mobile user, makes use of resources from distinct virtual organizations (i.e., the question of how to match ontologies from different virtual organizations is still an open issue for the current mobile grid approaches). There have been several attempts by researchers to address these factors of transparency in mobile grid environments. However, none of these studies has succeeded in taking all these factors into consideration in a single scheme, as shown in Table 1.

Table 1 shows that each author only considers two factors at most. For example, there is no approach that supports the management of interactive tasks (i.e., workflow and the disconnection of mobile devices) with the need to select resources in a semantic way to execute submitted applications from mobile devices simultaneously. In addition, the approaches which are concerned with disconnection do not support workflows. It should also be pointed out that none of the related works is concerned with the question of the battery consumption of mobile devices. All these factors are covered by our proposed approach.

Table 1 Approaches to mobile grid environments in other research studies

Related works	Resource selection in wired network	Disconnection	Workflow	Battery energy
Sajjad et al. [31]	No	No	No	No
Brooke et al. [1]	No	No	Yes	No
Chunlin et al. [5]	No	No	No	No
Hummel et al. [14]	No	No	Yes	No
Imran et al. [16]	No	Yes	No	No
Khalaj et al. [17]	No	Yes	No	No
Grabowski et al. [12]	Yes	No	No	No
Coronato et al. [6]	Yes	No	No	No
Wei et al. [33]	Yes	No	No	No

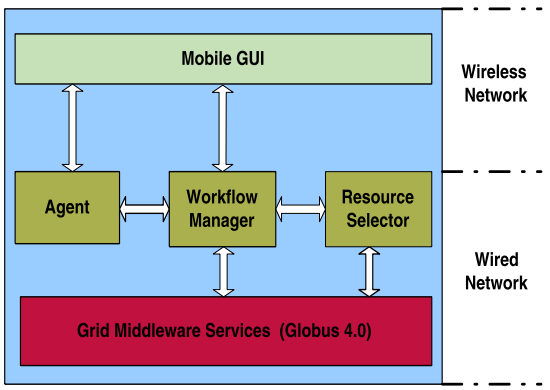

Fig. 1 The MITSIO—proposed architecture

3 The MITSIO—architecture prototype

In this section, there is a description of the MITSIO architecture prototype and its design characteristics and implementation. The proposed architecture is an implemented prototype that was developed by means of Java technologies that employed various Java packages which will be described for each component. This architecture employs the bottom-up approach of integration testing, where the components are integrated from lowest-level components to highest-level components in an incremental way. The main goal of the proposed architecture is to provide a greater degree of transparency for mobile users when using the mobile grid configurations. As a result, it conceals of the details of grid middleware such as workflow management (submission, execution, and monitoring) and resource selection in different virtual organizations. Figure 1 illustrates the architecture that has been created to build the prototype. It consists of four main components: *Mobile GUI*, *Workflow Manager*, *Agent*, and *Resource Selector*. On the one hand, the *Mobile GUI* component is specifically designed to wireless devices and, therefore, it deals mainly with factors of transparency for the wireless networks. On the other hand, *Resource Selector* concerns about the transparency in the wired network specially. While the *Workflow Manager* and *Agent* components handle with factors from both wired and wireless network

(as is shown in Fig. 1). In this way, MITSIO covers several factors of transparency from both wired and wireless networks, every factor which is supported by MITSIO will be discussed in the next subsections.

The components of the proposed architecture are as follows:

- *Mobile GUI*: this shows the mobile-adapted interfaces of unique access to the grid.
- *Workflow Manager*: this processes and manages requests (e.g., submission, monitoring, and downloading of application results) coming from mobile devices to be carried out in a grid environment. This component enables applications to be executed where distributed resources can be coordinated in multiple virtual organizations. It also acquires specific processing capacities through the integration of multiple teams involved in different parts of the application, and thus bringing about interorganizational collaborations;
- *Agent*: this adapts the execution flow to ensure the consistency of the application in a disconnection;
- *Resource Selector*: this is responsible for gathering information found in the grid environment and selecting resources in a semantic way through multiple ontologies (each from a different VO).

3.1 Mobile GUI

This component was designed and developed for interfaces that provide unique access to the grid, especially for cell phones and PDAs. Thus, we use tools and communication protocols that provide portability for the different devices. This component was implemented by means of Java technologies, for example, J2ME (*Java 2 Micro Edition*) *Wireless Toolkit* [37] and HTTP communication protocol, which provide portability and support a large number of devices. Despite the fact that *Mobile GUI* focuses on mobile devices, this component is based on Java technologies that have developed interfaces for clients and servers machines and for this reason, it can be easily extended to desktop machines. The *Mobile GUI* possesses the following functionalities: application submission; visualization of final and partial results of the optimized application, the only parts of the resultant files that are considered to be relevant for the user and that are loaded in the interface of mobile devices and lastly, the capacity to monitor the execution of the application, following the progress of the execution through the status of each task.

The proposed architecture provides an interface with the same functionalities for PDAs and cell phones. Thus, it allows the user flexibility in choosing which device to use, depending on his/her different needs. In other words, users can make use of the advantages offered by both PDAs (for example, less resource than cellular limitations and the fact that

Fig. 2 Application monitoring interface

no financial cost is incurred for data transmission) or/and cell phones (for example, lower acquisition cost and a bigger covering ray). It should be stressed that some of these interfaces are adapted in accordance with the restrictions imposed by different devices. The challenges posed by these adaptations are discussed in this section.

It has become necessary to provide alternatives to users to allow them to have a knowledge of the progress of its execution from different localities, since some application executions can take hours (or even days) to complete [32]. For this reason, the application monitoring could be a useful functionality for mobile users of the grid and cloud, because these devices provide a display in which users can check the execution progress of the workflow that is being made in wherever and whenever. This is because these devices allowed better portability than desktop computers, due to their weightlessness and small size.

Our approach provides a monitoring interface which complies with the screen restrictions of each device. Thus, with the interface of the *Mobile GUI*, the monitoring is initiated without the need for any interaction (or intervention) on the part of the user. The monitoring was developed to provide an enhanced capability to follow the execution of each application task. This ability is made possible by using the updated information of the status and adapted interfaces for the PDAs and cellular. Figure 2 illustrates the monitoring interface.

We implemented a PDA interface that represents the workflow of the application graphically in a single structure and provides an organized way to track the status of tasks in the interface. Moreover, it allows the execution of the several application tasks to follow simultaneously. Of the choice shown in [39], the non-DAG formalism of the workflow representation was most ideally suited to our approach, because it is flexible, and allows a representation of all the definition structures (such as, sequential, loops, and parallelism) as well as those contained in other Formalist options. As a result, our interface is able to represent workflows with complex structures and relationships. In addition,

it also provides a graphic representation that offers saving spaces in the screen of the devices. In other words, it employs the minimum number of graphic elements where circles (or nodes) indicate tasks and arches indicate transitions or dependencies among these tasks. In addition, the PDA monitoring interface has a range of standard colors in each circle to indicate the updated status of each task. Thus, when the monitoring interface requests information from the status, it translates this information into colors that fill the corresponding circles which represent the tasks.

The cell phone GUI has the same functionalities that are contained in PDA GUI. However, the monitoring interface is displayed in a different way in the cell phone. Moreover, the KVM (Kilobyte Virtual Machine) of some cell phones has not provided support to some classes and methods used to design the structure of the application workflow in the interface of the PDA (including the cell phone used in our architecture). As a result, it was necessary to show the status of the tasks in another way. In the case of the cell phone, the monitoring interface of the workflow displays the status of tasks in a form with text fields for each task, as shown in the right-hand side of Fig. 2. Instead of a color standard, the notifications of messages on the screen and the emission of sounds was used to indicate the tasks status in the cell phone.

3.2 Workflow manager

There are several proposals that are only able to submit and monitor one task at a time from a specific device in the mobile grid [5, 6, 12, 16, 17, 31, 33]. This means that users may submit tasks needed for solving a problem in an order that is inappropriate. Delays may also occur between the task submissions owing to a need for retransmissions caused by the high rate of errors of the wireless networks. For this reason, it is useful to have a mechanism in which the mobile devices can show an automated and coordinated facility to manage the flow of tasks in grid configurations in a transparent way. To achieve this, this article offers the *Workflow Manager* component by means of which mobile devices are able to submit applications to grid environments based on the workflow concept. In this context, this concept was conceived and implemented both to provide transparency and to submit and monitor applications in a coordinated and automatic way.

As well as being able to process and control requests coming from mobile devices, and to carry out in a grid environment, this component also collects information related to task execution, and performs all these functionalities for mobile users in a transparent way. Finally, it provides an automatic feature for mobile users, by dispatching all the tasks to available grid resources without any need for a user's interaction. In this way, it can show more agility in the execution of collaborative tasks to solve a problem. This component comprises three modules, which are as follows: *Controller*, *Engine,* and *Collector*.

Each submitted application has its *Engine* instance, *Collector* instance, and a unique Identifier. The Java CoG Kit package [21] was used for enabling this component to interact with Globus 4.0 [9]. The *Controller* module is responsible for receiving requests that arrive from mobile devices and ordering these requests, or commanding and controlling their order, through an Identifier. When this module receives a submission request, it creates an instance of the *Engine* module and directs the request to this instance, as is shown in Fig. 3.

The *Engine* is the main module of this component. This module sends a resource request (i.e., query) from an application to the *Resource Selector*, as it is presented in Fig. 3. Subsequently, the *Resource Selector* returns a list of machines to the *Engine* that contains the requirements of all the resources (e.g., hardware, software, and databases) for the execution of each application task. Following this, the *Engine* module specifies machines that are selected in the workflow definition file of the submitted application, and interprets this definition file (or workflow script). Definition files are described in the *Karajan* workflow language [20] of the *Java CoG Kit* package.

These scripts specify requested software, databases, and machines as well controlling the flow of tasks of each application. In addition, this module can interpret different workflow scripts to find out how they it must be submitted and controlled in the execution of each task flow of the different applications. In this component, each application has defined its proper workflow script for specifying its task flow. As a result, our architecture enables different applications to be carried out. Thereafter, the submission of each task is undertaken through the Grid Resource Allocation and Management (GRAM) grid service [9].

Finally, while these tasks are being submitted, their status is defined through events that take place during their execution. Thus, the *Engine* creates an instance, described as the *Collector*, for obtaining this information and for storing it in checkpoint files (i.e., XML files) that record the status in which each task is currently found (as is shown in Fig. 3).

3.3 Agent

Mobile devices are very susceptible to disconnections; there are two kinds of disconnection (i.e., involuntary and voluntary). The former can occur as a result of their limited resources and mobile nature. For example, involuntary disconnections can be caused by a reduction in the lifetime of the battery or interference in the wireless network. The latter may occur when the user is busy and decides to turn off the device (e.g., meeting or driving). The occurrence of a voluntary (or involuntary) disconnection of the mobile device

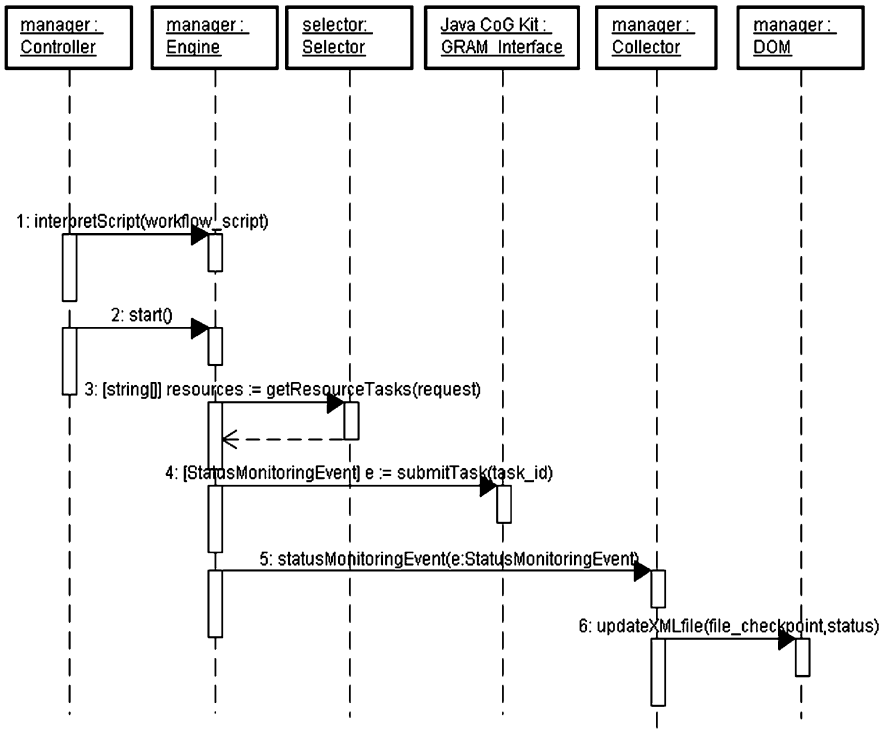

Fig. 3 Workflow manager sequence diagram

may be due to a fault. This fault can cause an error that can generate a failure. Thus, a fault can be defined as an interruption of the connection with the device, whereas an error refers to the impossibility of receiving information from a mobile user (e.g., input parameters in a specific task on the basis of the results obtained in previous tasks, or even the reference data stored in the device), and this error can generate a failure state, i.e., an incorrect result of the execution flow of application. This means that the transparency would improve if the approaches supported an adaptive mechanism for the flow of execution in the event of disconnection. It is important to be aware that these problems occur in the mobile environment while the workflows application is being executed. For this reason, the architecture includes an *Agent* component that provides the adaptive and optimized forms that are needed for a better adjustment of the workflow execution when an involuntary or a voluntary disconnection is detected.

The *Agent* component adapts the execution flow to ensure that the application is consistent in a personalized way for the user, in the event of disconnection. As a result, the *Agent* component checks the environment and adapts the execution flow of the submitted workflow application from the device. Thus, it was possible to detect the disconnection and adjust the flow, and in this way, prevent a failure state. The interactions between the *Workflow Manager* and *Agent* components are shown in Fig. 4.

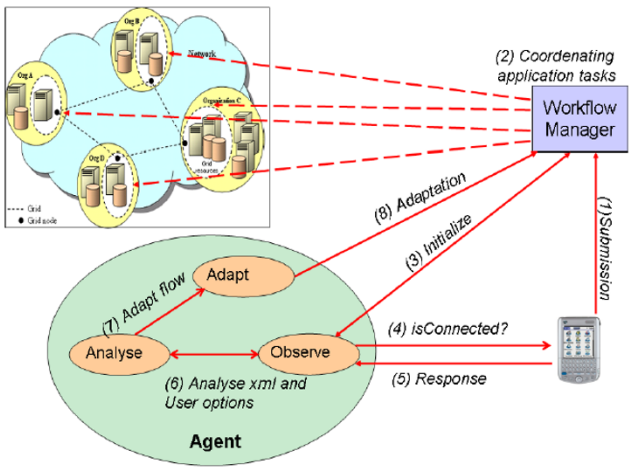

Fig. 4 Workflow manager and agent interactions

First, a user submits an application to the *Workflow Manager* (Step (1)). This module coordinates and manages the application tasks (e.g., an application with four tasks), i.e., controls the execution flow, in accordance with the workflow definition file (Step (2)). Next, it dispatches each task to the grid resource that has been selected. After this, the *Manager* creates an instance of the *Agent* (Step (3)) that will check, within a determined time interval, if the device that sent the submission request is connected (Steps (4) and (5)). The *Agent* continues the checking up to the moment

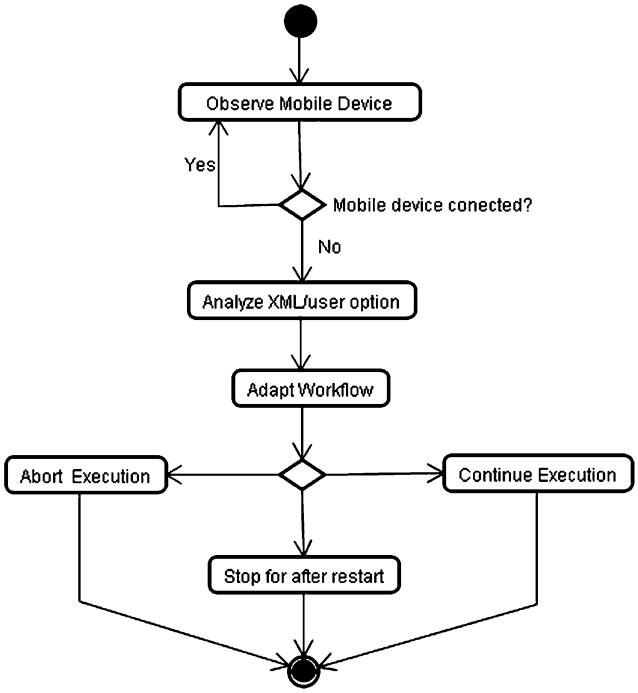

Fig. 5 Agent activity diagram

that the submitted application has completed its execution. If the *Agent* does not receive a response from the device in a specific time interval, a disconnection takes place and the execution flow is adapted (Steps (6), (7), and (8)). Thus, the Agent module has three basic operations: *Observe*, *Analyze,* and *Adapt*. These operations are outlined in Fig. 5.

The purpose of the *Observe* operation is to check the connection status with the device. This function is performed by means of a thread that was created for each device connected in this component. The thread awaits successive communications from the device. If the thread does not receive a return in a given time interval, the device is considered to be disconnected and then the *Analyze* operation is activated. For instance, if an application task needs data from the mobile user, the *Workflow Manager* stops the application execution and waits for these data during the defined time interval. If the data do not arrive during this interval, an exception is generated and the *Analyze* operation is activated.

The *Analyze* operation examines the requirements of the application (i.e., if the application is dependent or non-dependent) and decides how it must be processed in this situation. Another useful aspect of the proposal refers to the fact that a user can define the options (e.g., whether to continue, stop, or abort the application), when a disconnection occurs during an application execution and in situations where an application does not depend on the mobile user. This means that a user can decide to halt the application until the connection restarts. If the application has tasks involving dependence, the user can only define the options *Stopping for later restart* or *Aborting* in the execution of the specific task,

Table 2 Options for the decision of the analyzer

Application requirement/user options	Dependent application	Non-dependent application
Aborting	Yes	Yes
Continuing	No	Yes
Stopping for later restart	Yes	Yes
Default	Stopping	Continuing

i.e., the option *Continuing* is not allowed in this case. This feature makes it easy for the architecture to deal with the disconnection and, if necessary, adapt the execution flow, while taking into account the nature of the application and the options that are defined by the users. In other words, this characteristic allows a user to have a personal profile, so that there can be a personalized and transparent application execution when the disconnection is detected. Table 2 shows the options that the mobile user can define if there is a disconnection of the mobile device. Moreover, it should be pointed out that there is a specific situation regarding the dependent applications when the mobile device is on and connected, but the user is at a meeting or otherwise occupied and is thus unable to interact with the workflow application. In this particular situation, the mobile device is considered to be connected, although if the application is dependent, the execution will be stopped at the point in the workflow that requires the interaction of a user.

After making the definition of the *Analyze* operation, the *Adapt* is responsible for any modifications, when necessary. There are four possible situations where intervention is needed in the execution flow: aborting, stopping for a later restart, allowing the application to continue executing, and default. In these cases, the *Adapt* operation communicates with the *Engine* module to proceed with adaptations. If the *Analyze* operation finds that the application can continue the execution, the results will be stored so that they can be delivered to the user later.

When the *Adapt* operation defines that the application must be stopped to restart later, the *Workflow Manager* will store the states of the execution in the checkpoint file. Thus, when a device reconnects to the *Workflow Manager* component, it will be checked to determine if any task has not yet been completed. If this is the case, the user will be notified by the interface in *Mobile GUI* that there is a submitted application where the execution was not fully carried out. If the user confirms the restart of this application, the tasks states will be recovered and the application will restart the execution from the point where it had stopped.

In addition, as the wireless network has an unstable characteristic and grids and clouds are dynamic environments, it is difficult to guarantee that grid resources will be available during the disconnection period as well to foresee when the

Table 3 Hardware and software configurations of the implemented prototype

Nodes	Model	Processor	Memory	Operating system
PDA	Palm Tungsten C	400 MHz	64 MB	Palm OS 5.2.1
Server	AMD Duron	1700 MHz	512 MB	Linux Red Hat 9.0

device will be reconnected. The amount of time this disconnection might last can vary considerably for different reasons, such as the interference in the wireless network, the fact that the battery has reached the end of its lifetime, or the fact that the user is outside of the covering ray of the access point for an indeterminate period of time. As a result, *Workflow Manager* will make new resource requests to the *Resource Selector* when the device reconnects, and select new resources for the application tasks that have not completed their executions (e.g., this only applies when the user defines that the application must stop in a case of disconnection).

3.3.1 *Case study (A)—battery power consumption in the PDA*

The MITSIO architecture is tested in a mobile grid scenario prototype. In this scenario, there is a wired network where the resources reside and the mobile users access the grid environment through the access point (Wi-Fi 802.11b). PDAs are used as client nodes and the *Server* is the machine that is installed and configured with the grid middleware Globus 4.0 [9]. Table 3 shows the hardware and software configurations of the implemented prototype.

In both case studies, an application was used that was developed by the Genome Project [23] was used. This application-based workflow aims to perform a DNA sequencing of the *Gluconacetobacter diazotrophicus* bacterium [23]. This scientific grid workflow enables to describe the capacities of all the components of the proposed architecture. The scientific application comprises seven tasks. The detailed description of each task is visualized in [23].

The objectives of this case study are to evaluate the battery consumption of PDA from two distinct perspectives. The first is to evaluate the MITSIO architecture against the approaches that fail to support the workflow in managing the interactive application tasks. The second is to provide evidence of the impact of the overhead messages generated by the *Agent* component on the battery consumption of the PDAs (i.e., the *Observe* operation), since the *Agent* sends continuous messages to ensure that the mobile device is still connected. It is well known that a reduction in the access to wireless network can decrease the level of the dissipation of the battery energy of these devices, i.e., increase its

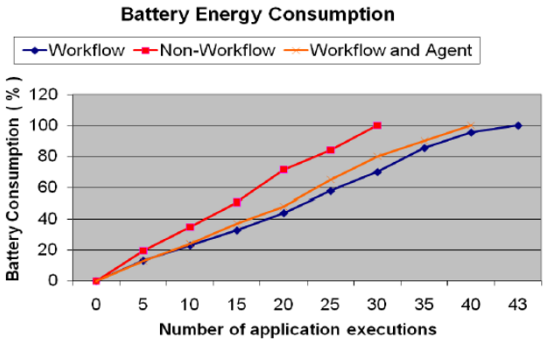

Fig. 6 Battery usage

lifetime. Extending the battery lifetime is one of the most critical and challenging problems in these devices [24, 30]. An approach that provides automation for submissions of application, which occurs in our architecture, can reduce the rate of battery consumption in the mobile devices. MITSIO sends fewer submission requests, and thus is able to reduce the dissipation of battery energy of these devices.

We define the compared approaches on the graph as follows: *Workflow* is the approach that is able to submit and monitor the application by making a single submission request. This is because of the *Workflow Manager* component which controls the execution flow and invokes the necessary computational tools without requesting the mobile users to perform all the steps involved in the workflow submission and monitoring. Hence, the workflow submission only has to send one message. This is a differential aspect when compared to other related works, described as *Non-Workflow* in [5, 6, 12, 16, 17, 31, 33]. In these other approaches, a user controls the submission sequence of each task of the workflow application that collaborates to solve the same problem. Furthermore, in the *Non-Workflow* approach, a user submits each task through the interface. Hence, every task of the submitted workflow requires a message, and thus consumes more battery energy from the mobile devices than it does in the *Workflow* approach. Finally, the *Workflow and Agent* approach is in fact the proposed architecture adopted in this paper; it contains the *Agent* component and can thus adjust the flow of execution when the disconnection is detected.

In the graph of Fig. 6, there is a comparison of the mean battery power consumption for submitting and monitoring applications for the resolution of a problem. This contrasts the use of the *Workflow* approach with the *Non-Workflow* approach implemented in [5, 6, 12, 16, 17, 31, 33]. The result shows that *Workflow* achieved an average saving of 31% when compared to the *Non-Workflow* approach. The *Workflow* approach also allows a larger number of executions to be made, i.e., on average 12 more application executions than the *Non-Workflow* approach, as can be seen in the graph.

Table 4 Average and confidence interval of compared approaches

Approaches	Average	Confidence interval
Non-workflow	16.67%	(+/−) 2.0758%
Workflow	11%	(+/−) 2.4651%
Workflow and agent	12.85%	(+/−) 1.8360%

Twenty-four experiments were conducted on each approach and each experiment involved measuring the percentage of the battery consumption in five application executions of the sequencing problem. In each application execution, the consumption percentage for submitting and monitoring each task in the *Non-Workflow* approach was calculated, as well as the consumption percentage for submitting and monitoring the application execution in the *Workflow* approach. In addition, in each experiment, the resources were entirely devoted to the tests and the time interval for sending notifications was found to be equal in both approaches. This suggests that these variables do not influence the results of each approach.

Table 4 shows the average and confidence interval (95% of significance level) of battery consumption for each experiment. The results provide evidence of a significant difference between the *Workflow* and *Non-Workflow*. Furthermore, despite the overhead caused by the *Agent* component, the *Workflow and Agent* approach still lead to a significant reduction in battery consumption when compared with the Non-Workflow approach.

3.4 Resource selector

The difficulty of establishing agreements in the customary terms to characterize resources and requests should be underlined, particularly with regard to the selection of resources in the wired network, since different virtual organizations (VO) have their own autonomy when defining their resources in different forms (VO can play the role of infrastructure providers in cloud environments [40]). In others words, these resources can be based on syntactically distinct descriptions, while sharing the same semantics. In a classic example, two distinct organizations refer to memory size characteristics as *physical_memory_size* and *main_memory_size*, respectively. Given the fact that one query for resources expresses a wish for a resource that has *physical_memory_size* >= 1024 MB, only those resources that satisfy this restriction and that employ the term *physical_memory_size* will be returned.

As [40] shows, there is still a lack of cloud architectures that can manage heterogeneous resources. In the light of this context, it would be useful to employ a mechanism for resource selection in the grid and cloud that is flexible, extensible and transparent. In other words, the mechanism should not only take account of the syntax of the resource descriptions, but also the meaning involved in these descriptions, while at the same time, it conceals the syntax of description as well as the virtual organization which the selected resources belong to. Furthermore, the distinct description of other resources should also be taken into account such as, software (e.g., applications) and platforms (e.g., databases). For this reason, the proposed architecture also includes the *Resource Selector* component in this section, which chooses the appropriate resources (i.e., software and platform) for each submitted task of the application.

The *Resource Selector* component takes account of different forms, like the grid and cloud resources that have been described, using a selected semantic integration of multiple ontologies. The concept of ontologies provides flexible and expansible structures that make interoperability possible in different areas of knowledge. In addition, it enables interaction to occur between people of different cultures and vocabularies, and in common situations found in heterogeneous and distributed environments such as cloud and grid computing. In the context of the mobile grid, this issue of resource selection is overlooked by the related work [6, 12, 33] on resource selection in the wired network. Hence, this component was designed and implemented by means of semantic web technologies to provide a resource matching system based on the semantic integration of multiple ontologies (this is illustrated in Fig. 7). All the resources are described by means of different ontologies that supply the application requirements. We adopt the meaning that is denoted by the terms used to describe the resources and not by simply examining the syntax.

The Jena framework [3] was employed, because it is an easy way to allow semantic web applications to be created with the aid of Java language. In addition, this framework has useful features such as the following: it manipulates OWL ontologies and is capable of inferring information from the modeled knowledge in the ontologies that use an inference engine (i.e., are based on rules). The ontologies examined in this article were developed by means of the Protege editor [18].

In Fig. 7, there is a sketch of the grid resource matching selector component. The three main modules of the component have the following characteristics:

– *Ontologies Integration Portal*: This module represents the interface used by a VO to bring about the semantic integration of its own ontology within the matching system. The ontology developer is responsible for the description of the resources of a VO. By adopting the concept of a Reference Ontology (RO), it can persuade the developer to regard semantic equivalences in terms of its VO ontology and the RO. Moreover, this interface was not prepared to allow the mobile users to integrate the multiple ontologies, i.e., define equivalences. Only specialists (the

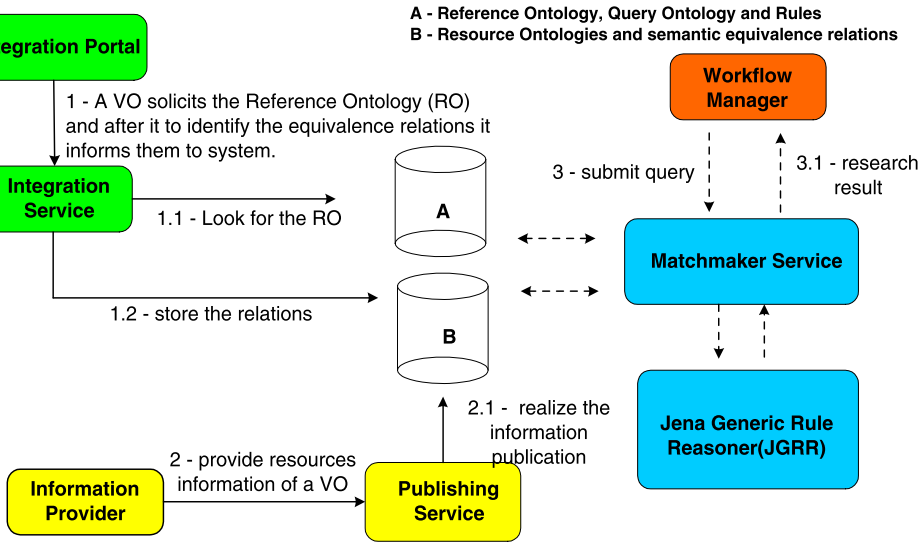

Fig. 7 Component resource selector

ontology developers of each VO) and those who have the knowledge necessary for interacting with knowledge systems are able to relate the equivalences that can be found among the ontologies for the *Resource Selector* component. As a result, this interface was only developed on desktop computers, which enabled it to be less restricted and more flexible for this functionality;

– *Information Providers*: This module represents the resource information collectors of the VO. Each collector publishes resource information from its own system, through Publication Service, as is shown in Fig. 7;
– *Matchmaker*: The semantic matching operation is the main function of this component. The operation takes account of the semantic equivalences that come from the semantic integration of the ontologies and also the restrictions defined in the queries that are submitted by the *Workflow Manager*.

A more detailed description of the proposed ontologies and portal can be found in [35].

3.4.1 The proposals for ontologies

Ontology can be defined as a formal and explicit specification of a shared concept [36]. A differential area where ontologies can be applied is in the integration of systems and databases [10]. The information exchange in execution time is an important function, which is also known as interoperability. The mapping of ontologies is necessary for interoperability to occur between ontologies. In other words, the objective is to find semantic correspondences between the terms that are defined in these ontologies. Studies show that the automatic tools employed for the mapping between ontologies do not recognize all (or the majority set of) semantic equivalences between the terms, which thus reduces the degree of interoperability between the systems that use these ontologies. This fact suggests that human interaction is still necessary [10, 25].

There are three different approaches for the integration of information sources based on ontologies [10] that can be defined as follows: global ontology, multiple ontology, and the hybrid approach. As stated in [4], on the basis of an analysis of these three approaches, it is possible to conclude that the hybrid approach has important characteristics, because it does not restrict the diversity of models that describe a single domain, since each model has its own particular ontology. In addition, this approach does not involve a complex procedure of ontology mapping, because the terms defined in these different ontologies are shared semantically with those of a global ontology. Our reason for deciding to adopt the hybrid approach in our architecture was based on the advantages that this method provided.

The integration of different ontologies from a different VO in our research work was achieved by sharing a common ontology. In [38], there is a closed approach, known as Core Grid Ontology (CGO) that is a high-level framework, where grid domain experts represent all the grid concepts semantically in a coherent and consistent form. CGO, with the support of OWL, can grasp and model concepts and basic grid knowledge such as *GridResource*, *GridMiddleware*, *GridService,* and *GridApplication*.

We decided to design and implement two ontologies. These ontologies were called *Reference Ontology* (RO) and *Query Ontology* (QO). RO is the common ontology used by specialists to accomplish the semantic integration of ontologies from distinct VO within this resource matching component. In other words, the specialist of each VO defines the

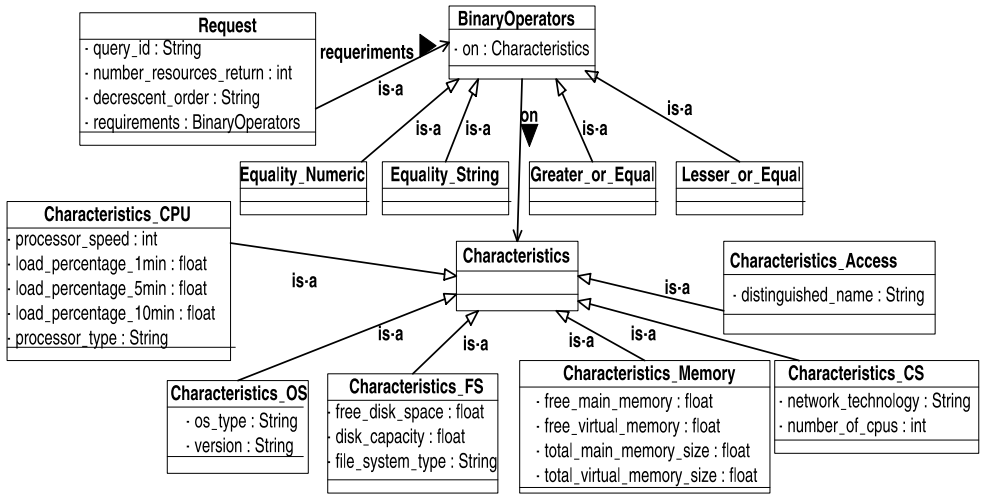

Fig. 8 The query ontology proposal

equivalence relations based on the RO. In addition, RO can be understood as being an enhanced proposal, because it has more resource details when compared to the CGO (examples of this are: its memory capacity, and the operation system version). The CIM model [7] was used to describe the elements that compose computing resources that are represented by the direct and indirect subclasses of the *ComputingResourceElements* superclass.

Figure 8 shows the QO through a class diagram. The QO was built as an ancillary query language that is recognized by the developed *Matchmaker*. The QO employs terms that were previously defined in the RO, and which denote the characteristics of its resources.

Common restrictions were conceived to help specialists to describe resource requirements precisely. These concepts were created by means of the *BinaryOperator* class. In other words, these operators suggest to the *Matchmaker* module how to compare the values that are attributed to the terms defined in QO and the values attributed to the terms that represent the resource characteristics defined in the different ontologies of VO.

The following properties were defined within the *Request* class, as shown in Fig. 8:

– *query_id*: query identifier;
– *decrescent_order*: a directive that instructs the *Matchmaker* about which order criterion (decrescent order) of the resources will be returned. This instruction is based on the resource characteristics (numeric value) that are defined within the subclasses of *Characteristics* class;
– *number_resources_return*: this directive specifies the maximum number of resources the *Matchmaker* should return;
– *requirements*: the property that establishes a relation between *Request* class and *BinaryOperators* class instances. This relation defines the restrictions that exist inside a query.

The *BinaryOperators* class has the property *on* which connects the operators to the characteristics from the resoThis property characterizes the restrictions that the resources should match. The subclasses from the *Characteristics* class consist of the specialized characteristics from the resources for providing more accurate information to each type of resources. Examples of these are *Characteristics_CPU*, and *Characteristics_OS* classes which aggregate the characteristics of the following elements that compose the computational resources: processors and operating system.

3.4.2 Matchmaker

This module, as has been previously mentioned, has the function of carrying out semantic matching between the published resources and queries. In other words, the *Matchmaker* is responsible for matching available resources and queries based on the restrictions defined in the queries and semantic equivalence relations found in the integration process of the ontologies. The following steps illustrate how the matching operation is put into effect:

– *Verification of the Query Consistency*: this phase consists of checking the queries for inconsistencies. Thus, this element avoids unnecessary processing of queries. Inconsistencies are detected through rules that rely on the restrictions described in the queries, in the knowledge structure and information represented in the RO. An example of an inconsistent query is shown in Table 5.

Table 5 shows a query (request) of resources that could fulfill all the following requirements: Windows XP operating system, EXT3 file system and number of pro-

Table 5 Example of an inconsistence query

Requirement	Value
Request.query_id	'query_1'
⟨X⟩.Characteristics_OS.os_type	'Windows XP'
⟨X⟩.Characteristics_FS.file_system_type	'EXT3'
⟨X⟩.Characteristics_CS.number_of_cpus	'4'

cessors equal to 4 (the binary operator used was *Equality_String*). The expression ⟨X⟩, that appears in Table 5, represents the following path in the QO structure "Request.Equality_String." It is clear that there are two inconsistencies. There is no operating system, from the Windows family, to support the EXT3 file system. In addition, the chosen operator to compare the value assigned to the *number_of_cpus* is incorrect. This operator is used to compare the character sequence and not the integer numbers.

The *Matchmaker* uses the JGRR reasoner to evaluate the rules in the queries sent by the *Workflow Manager* component, as shown in Fig. 7 shows. Each created and stored rule in the system has two parts. The first conceives a set of premises that express a possible inconsistency in the query. The second part is the conclusion of the rule. The conclusion consists of a message that describes the inconsistency expressed by the premises. When all the premises have been evaluated as true by the JGRR on the query, a message is returned informing the inconsistency found in the query. As a result, if any inconsistency is discovered during this phase, it will not return any result to the *Workflow Manager* component. Before making the workflow application available for submission and monitoring through the *Mobile GUI* component, it is important to stress that the workflow script which meets the requirements of all the resources (i.e., computers, software, and databases) used in the workflow execution is validated by the *Resource Selector* component. In other words, it is tested to check if there are any inconsistencies. This is also done by the specialists, and hence it is also transparent for the mobile user.

– *Query Expansion*: This phase deals with the query expansion which is based on the knowledge structure and information represented in the RO. An interesting example that illustrates how this module works can be verified when an operating system equal to Unix is requested. In an ordinary matching system, only resources with Unix will be returned to a user. In our system, because the concept of an operating system as a more comprehensive class has been represented in the RO, there will be more options to choose from. The *OperatingSystem* class considers Windows, Unix, Linux, MacOS as possible operating systems, as is shown in Fig. 10. Table 6 provides instances that were created to represent these types of operating systems.

Table 6 Unix and Linux classes instances defined in RO

Class	Class instances
Unix	SunOS, AIX and FreeBSD
Linux	Debian, Slackware and Fedora Core

```
- <queries_extension_rules>

<rule> [ GetOSsLikeUNIX: (?A greq:os_type ?SO) (?B rdf:type owl:Class) equalURI(?B 'Unix')
        equalURI(?B ?SO) (?D rdf:type ?B) (?D ont_ref:os_type ?SO1) (?E rdfs:subClassOf ?B)
        equalURI(?E 'Linux') (?F rdf:type ?E) (?F ont_ref:os_type ?SO2) ->(?G gres:result_value ?SO1)
        (?H gres:result_value ?SO2) (?I gres:result_value 'Unix') (?F gres:result_value 'Linux') ]
</rule>
....
```

Fig. 9 Example of a query extension rule

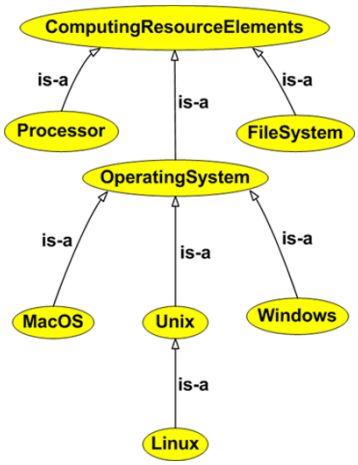

Fig. 10 Part of reference ontology developed

Through the rules, it is possible to locate the knowledge and representation of the instances that exist inside the RO. Thus, it is possible to expand the queries (as Fig. 9 shows). By employing this rule (GetOSsLikeUNIX), the *Matchmaker* module will return not only Unix, but also the SunOS, AIX, and FreeBSD resources.

In addition, the *Matchmaker* returns resources such as Debian, Slackware, and Fedora Core. The subclass relation between Linux and Unix classes is transitive. In other words, this means that through this relation, it was possible to express the knowledge that Linux operating systems are also regarded as a type of Unix system.

– *Resources Search*: After conducting the query consistency verification and its expansion, the *Matchmaker* attempts to find out which resources satisfy the restrictions defined in the queries for the *Workflow Manager*. A clearer visualization of this step is exemplified in

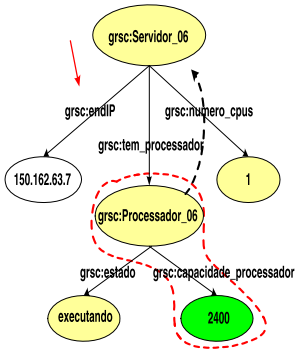

Fig. 11 Resource describes in ontology A

Fig. 11. In this figure, a RDF graph represents a computational resource and is based on the assumption. For example, if a user has defined in a query the following restriction *processor_capacity* >= 1800 MHz, this means that the resources that have to be returned must have at least 1800 MHz of processing capacity. The *Matchmaker* first searches in the equivalence relations stored in the system to find out which terms are equivalents to the term *processor_capacity*. In ontology A, the term equivalent is *capacidade_processador*. This term represents the predicate of the triple, that we denominate triple base, and is shown in the red-traced line in Fig. 11. The procedure will determine if the value 2400 MHz (attributed to *capacidade_processador*) satisfies this condition. If this value does satisfy it, the next step will consist of a search for the triple from the RDF graph, which represents the computational resource that satisfies the restriction. The triple which allows the predicate to denote the IP address identifies the desired resource. We chose the IP address since it is a single qualifier of a selected resource in distributed environments.

The *Matchmaker* initiates a search for the subject *grsc:Processador_06* of the triple base, because the predicate of this triple has the feature that satisfies the restriction. The *Matchmaker* checks if this subject has the predicate that means the IP address. However, as the subject *grsc:Processador_06* does not show this predicate, the *Matchmaker* checks between triples which have this predicate. The triple <*grsc:Servidor_06, grsc:tem_processador, grsc:Processador_06*> is the only one that has *grsc:Processador_06* an object and thus the *Matchmaker* has the predicate that denotes the IP address. Based on the semantic integration of ontology A with the RO, Matchmaker can check whether the term endIP (indicated by the red arrow in Fig. 11) expresses the characteristic IP address in ontology A. In this way, the *Matchmaker* stores the triple which indicates the resource that satisfies the restriction of the query.

For each existing restriction in the query, the *Matchmaker* searches the triples that represent the resources that satisfy the restriction. These triples are stored in a specific set. This procedure occurs for all the restrictions defined in the query. The next step involves completing the intersection of these sets to determine which resources satisfy all the restrictions, thus forming what we call the solution set. If the query has directives, these are applied to the solution set and the result is returned to the *Workflow Manager*. Otherwise, the solution set itself is returned to the *Workflow Manager*.

The *Resource Selector* component has its own autonomy to describe the resources. As a result, this component provides a semantic resource matching process which enables it to gather the resources required to integrate ontologies from a different VO and define an ontology to serve as a query language.

3.4.3 Selector characteristics

With regard to this component shown in Fig. 7, based on the semantic integration for diverse ontologies, has the following characteristics:

- *Ontologies Mapping*: The semantic equivalences are established between the terms from the resource ontology of a specific VO and the shared RO. Equivalences are binary relations between concepts, or between properties that characterize these concepts. These relations are recognized and informed by developers from the VO ontologies, during the integration process of their ontologies in the implemented prototype;
- *Verification of the Query Consistency*: The proposed semantic matching system adopts a meticulous checking process for queries. Inconsistencies found in the queries help prevent the Matchmaker from working unnecessarily on erroneous requests;
- *Query Expansion*: Query Expansion: a query can be semantically expanded to find more computational resources. For instance, a query looking for an AMD processor will return Opteron and Sempron processors;
- *Automatic Resource Search*: Creating a query language as an ontology allows us to build a query engine based on the restrictions defined in the queries and relations equivalences. It is possible for a specialist to navigate over different structures which describe computational resources, without knowing their formal representations. As a result, specialists can prepare effective queries in a more user-friendly fashion.

3.4.4 Case study (B)—the semantic integration of ontologies

This subsection describes a case study that provides evidence of the capacity of the *Resource Selector* component

to select resources from different virtual organizations based on the semantic integration of ontologies which describe their resources clearly. The workflow definition files (i.e., workflow scripts) are partially predefined by a specialist (i.e., grid manager) using the Karajan workflow language. This is possible because the specialists are likely to know some of the requirements of resources, for example, the features of computers, software and databases, for each workflow task before its execution. This means that the workflow definition file is formed in two phases, that are as follows: firstly, the specialist defines some of the requirements of the hardware (e.g., memory and operating systems), software and databases for the workflow. Following this, the *Resource Selector* component defines the workstations or machines (hardware) to run the application in accordance with these requirements. Hence, it is expected that after an application has been submitted from a mobile device, the *Resource Selector* component of the architecture will be able to find more suitable resources that can fulfill the requirements of all the application tasks. To reach this level, we gathered real machine information in a grid configuration and linked this information to its policies. In other words, all the ontologies were really published in the middleware grid of our prototype that was employed (i.e., the Monitoring and Discovery System (MDS) service in Globus 4). Furthermore, we used the same workflow application example employed in the case study of battery consumption.

The validation of the *Resource Selector* component was carried out by conducting experiments involving three ontologies, known as OV_1, OV_2, and OV_3 ontologies, taken from [19, 22, 29]. Using ontologies from other sources can give a better idea of how to show the different ways of describing resources in grid and cloud environments. A query for searching resources that meets the requirements necessary to carry out the application produced for each task is illustrated in Table 7. These queries were defined by using the QO and associated features in the application.

Each ontology represents a single VO which has its own resource ontology. The use of three ontologies shows the wide range of views that the resources can have in different organizations. The three virtual organizations had their resources described by ontologies developed in [22] (in the English language), [27] (in the Portuguese language) and [29] (in the Portuguese language), respectively (all of the three are designed basing on the OWL language). Each organization published 10 descriptions of resources, totaling thirty resources. The semantic equivalences were informed together with the resources established by each VO when integrating its ontology in the system. These equivalences are shown in Table 8. The first line in Table 8 shows the *os_type*, *SO* and *nomeSO* terms that were defined respectively in the ontologies OV_1, OV_2, and OV_3. These terms are equivalent to the term *os_type* defined in the RO, and thus in the QO denoted operating system.

Table 7 Directive and restrictions of research defined for each task

Directives/Restrictions	T1	T2	T3	T4
JR.request_id	query_T1	query_T2	query_T3	query_T4
JR.owner	vinicius	vinicius	vinicius	vinicius
JR.decrescent_order	total_main_memory	-	processor_capacity	processor_capacity
JR.number_resources_return	1	1	1	1
JR.Equality_String. **os_type**	-	Unix	-	Unix
JR.Greater_or_Equal.**total_main_memory**	>= 1024 MB	-	-	-
JR.Greater_or_Equal.**processor_capacity**	-	-	>= 3000 MHz	>= 3000 MHz
JR.Equality_String. **software_id**	S1	S2	S3	S4
JR.Equality_String. **database_id**	-	-	-	-

Directives/Restrictions	T5	T6	T7
JR.request_id	query_T5	query_T6	query_T7
JR.owner	vinicius	vinicius	vinicius
JR.decrescent_order	-	total_main_memory	processor_capacity
JR.number_resources_return	1	1	1
JR.Equality_String. **os_type**	-	-	Unix
JR.Greater_or_Equal.**total_main_memory**	-	>= 2048 MB	-
JR.Greater_or_Equal.**processor_capacity**	-	-	>= 3000 MHz
JR.Equality_String. **software_id**	S5	S6	S7
JR.Equality_String. **database_id**	-	DB1	DB2

Table 8 Equivalence relations

Requests Ontology	OV_1 Ontology	OV_2 Ontology	OV_3 Ontology
os_type	os_type	SO	nomeSO
max_number_of_processes	max_number_of_processes	numero_maximo_processos	-
total_virtual_memory_size	swap_memory_size	total_swap_MB	-
login	login	nome_conta	nomeConta
number_of_processes	number_of_processes	-	-

The *Workflow Manager* component sends queries regarding the application to the *Resource Selector* (i.e., requests of resources). When the selector receives a submission from a mobile user who wishes to carry out this application, it checks which resources meet the needs of each query. This process of verification is based on the equivalence relations between the terms of the QO and integrated resource ontologies. For instance, task T7 shown in Table 7 needs a resource that can be accessed by the user *vinicius* (the *owner* term), which has *Unix* operating system (*os_type* term), at the least *3000 MHz* (*the processor_capacity* term) of clock processor speed. In addition, the *software* resource must have software *S7* (the *software_id* term) and the database *DB2* installed (the *database_id* term). Between the resources that have these characteristics, the Resource Selector must return those that have the largest processing capacity (the criterion attributed to the *decrescent_order* directive).

The resource (e), presented in Table 9, was the resource returned to the *Resource Selector* in accordance with query_T7. The (c) and (e) resources comply with the restrictions defined in query_T7 (requisite of task T7). However, the resource (e) was chosen, as it has a greater processing capacity, thus satisfying the classification criterion defined in query_T7 (*processor_capacity*). It is important to highlight both the semantic matching produced by the *Resource Selector*, among the terms that express the characteristics

Table 9 Resources that satisfy the tasks requirements

Resources Characteristics	Resource (a)
Unitary_Computer_System.ip_address	150.162.56.12
Unitary_Computer_System.authorized_account. **distinguished_name**	vinicius;caetano
Unitary_Computer_System.running_os. **os_type**	SunOS
Unitary_Computer_System.has_software. **software_id**	S3, S5, S7

Grid Resource (a) shared for the OV_1 that satisfies requirements of T2 and T5 tasks

Resources Characteristics	Resource (b)	Resource (c)
Host.endIP	140.68.107.10	140.68.87.50
Host.contas_autorizadas. **nome_conta**	parra, mario, vinicius, caetano	parra, alex, vinicius
Host.tipoSO. **SO**	Fedora Core	Fedora Core
Host. **total_memoria_MB**	16384.0 MB	3062.0 MB
Host. **velocidade_cpu**	2400.0 MHz	3000.0 MHz
Host.tem_software. **nome_software**	S1, S5, S6	S2, S7
Host.tem_base_dados. **nomeBD**	BD1, BD2	BD2

Grid Resources (b) and (c) shared for the OV_2 that satisfies requirements respectively of T1 and T6, and T7 tasks

Resources Characteristics	Resource (d)	Resource (e)
Host.ipHost	147.160.50.37	147.160.12.19
Host.permissoesUsuarios. **idConta**	anubis, vinicius, guilherme	vinicius, parra
Host.temSistemaOperacional. **nomeSO**	Debian	Debian
Host.temCPU. **clockCPU**	3200.0 MHz	3200.0 MHz
Host.temSoftware. **identificador_software**	S1, S3, S4	S1, S3, S4, S6, S7
Host.tem_base_dados. **identificador_base_dados**	-	BD1, BD2

Grid Resources (d) and (e) shared for the OV_3 that satisfies requirements of T3 and T4, and T7 tasks

of the resource (e) and those that were used in the query. The equivalence relations between the terms derived from the semantics integration of the OV_3 ontology and the RO are as follows: *idConta* and *owner*; *nomeSO* and *os_type*; *clockCPU* and *processor_capacity*; *identificador_software* and *software_id* and lastly, *identificador_base_dados* and *database_id*. In addition, it should be pointed out that the query extension enabled the *Resource Selector* to recognize that the resource (e) has a *Unix* operating system, which was formerly based on the RO and obtained through the rule that determines how the *Linux* (Debian) operating system is regarded to be a type of the *Unix* operating system. Hence, this case study provides evidence of the capacity of this component to choose a set of resources (hardware, software, and database) for every workflow task, by selecting these resources from distinct VOs. This is possible because of the semantic integration of multiple ontologies.

4 Conclusion and recommendations for further studies

In this article, we have outlined a proposal and a prototype implementation of an architecture, called MITSIO, to provide more complete transparency for mobile users when they require the processing capacity of the grid configuration to run applications. MITSIO can attain a good level of transparency for mobile users, since it supports several factors of transparency from both wireless and wired networks that no previous work has been able to offer in a single approach, such as management of interactive tasks (submission and monitoring of workflows), resource selection in the wired network and the management of inherent characteristics of mobile devices (disconnections and the problem of a reduced battery lifetime).

The case study suggests that MITSIO is able to select suitable resources from different VOs for the execution of applications submitted from mobile devices. It is worth noting that these executions took account of the different descriptions of these resources that can be found inside each VO. Finally, the experimental results suggested that the MITSIO led to a reduction in the consumption of the battery power of mobile devices for submitting applications, despite the overhead that was generated by the *Observer* operation.

MITSIO supports some of the features that help to provide an infrastructure for cloud computing such as, sharing resource pooling, managing heterogeneous resources, geo-distribution and ubiquitous access through mobile devices; self-organizing such as automated application execution through workflow as well the adaptation of the workflow execution when the wireless disconnection is detected. Furthermore, resource matching is based on the semantic integration of multiple ontologies that brings about transparency in the software and platform among the different infrastructure providers or VO.

As a means of advancing this research, an algorithm will also be implemented to provide a time interval for determining the disconnection state and monitoring workflow in a more dynamic fashion. In other words, there will be consideration of input parameters, e.g., battery lifetime and/or traffic volume in the wireless network. Moreover, the capacity to restrict queries will be extended by widening the QO. As a result other binary operators (e.g., different [! =] and multiplicity [∗]) will be expressed.

References

1. Brooke J, Parkin M (2005) A PDA client for the computational grid. In: WETICE'05. IEEE Computer Society, Washington, pp 325–330
2. Bruneo D, Scarpa M, Zaia A, Puliafito A (2003) Communication paradigms for mobile grid users. In: 3rd (CCGrid'03), pp 669–676
3. Carroll J, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K (2004) Jena: implementing the semantic web recommendations. In: 13th World Wide Web conference, pp 74–83
4. Casare S, Sichman JS (2005) Using a functional ontology of reputation to interoperate different agent reputation models. JBCS 11(2):19–94
5. Chunlin L, Layuan L (2011) An economics-based negotiation scheme among mobile devices in mobile grid. Comput Stand Interfaces 33(3):220–231
6. Coronato A, Pietro GD (2008) Mipeg: a middleware infrastructure for pervasive grids. Future Gener Comput Syst 24(1):17–29
7. DMTF (2007) Common information model (CIM) standards. Available: http://www.dmtf.org/standards/cim. Last access on 5th March 2011
8. Farooq U, Khalil W (2006) A generic mobility model for resource prediction in mobile grids. In: CTS. IEEE Computer Society, Washington, pp 189–193

9. Foster I (2005) Globus toolkit version 4: oftware for service-oriented systems. In: IFIP NPC'05, pp 2–13
10. Freitas F, Stuckenschmidt H, Noy NF (2005) Ontology issues and applications guest editors' introduction. JBCS 11(2):5–16
11. Gonzalez-Castano F, Vales-Alonso J, Livny M (2002) Condor grid computing from mobile handheld devices. ACM SIGMOBILE Mobile Comput Commun Rev 6(2):18–27
12. Grabowski P, Kurowski K, Nabrzyski J, Russell M (2006) Context sensitive mobile access to grid environments and VO workspaces. In: MDM. IEEE Computer Society, Washington, p 87
13. Hollingsworth D (1996) Workflow management coalition. reference model and API specification. WfMC-TC00-1003
14. Hummel KA, Bohs G, Brezany P, Janciak I (2006) Mobility extensions for knowledge discovery workflows in data mining grids. In: DEXA. IEEE Computer Society, Washington, pp 246–250
15. Hwang J, Aravamudham P (2004) Middleware services for P2P computing in wireless grid networks. IEEE Internet Comput 8(4):40–46
16. Imran N, Rao I, Lee YK, Lee S (2007) A proxy-based uncoordinated checkpointing scheme with pessimistic message logging for mobile grid systems. In: HPDC '07: Proceedings of the 16th international symposium on high performance distributed computing. ACM, New York, pp 237–238
17. Khalaj A, Lutfiyya H, Perry M (2010) The proxy-based mobile grid. In: Mobile wireless middleware, operating systems, and applications. Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering, vol 48. Springer, Berlin/Heidelberg, pp 59–69
18. Knublauch H, Musen M, Rector A (2004) Editing description logic ontologies with the protégé owl plugin. In: 17th international workshop on description logics
19. Kurkovsky S, Bhagyavati AR, Yang M (2004) Modeling a grid-based problem solving environment for mobile devices. ITCC'04 2(2):135–136
20. Laszewski G, Hategan M (2005) Workflow concepts of the Java cog kit. J Grid Comput 3(3–4):239–259
21. Laszewski GV, Foster I, Gawor J, Lane P (2001) A Java commodity grid kit. Concurr Comput: Pract Exp 13(8–9):643–662
22. Lee TB (2006) World wide web consortium (W3C). Web Page [Online]. Available in: http://www.w3.org/Consortium/. Last access on 20th May 2011
23. Lemos M (2004) Workflow para bioinformática. Ph.D. thesis, PUC-Rio, Brazil, Rio de Janeiro. www.inf.puc-rio.br/~melissa/publicacao/download/tese_melissa/Tese_Melissa_Lemos.pdf. Last access on 10th April 2011
24. Mohapatra S, Cornea R, Oh H, Lee K, Kim M, Dutt ND, Gupta R, Nicolau A, Shukla SK, Venkatasubramanian N (2005) A cross-layer approach for power-performance optimization in distributed mobile systems. In: IPDPS
25. Noy NF (2004) Semantic integration: a survey of ontology-based approaches. ACM SIGMOD Rec 33(4):65–70. Special Issue on Semantic Integration
26. Park SM, Ko YB, Kim JH (2003) Disconnected operation service in mobile grid computing. In: ICSOC'03. LNCS, vol 2910. Springer, Berlin, pp 499–513
27. Pernas A, Dantas MAR (2005) Grid computing environment using ontology based service. In: 5th ICCS'05. LNCS, vol 3516. Springer, Berlin, pp 858–861
28. Phan T, Huang L, Dulan C (2002) Challenge: integrating mobile wireless devices into the computational grid. In: Proceedings of the 8th annual international conference on mobile computing and networking, MobiCom '02. ACM, New York, pp 271–278
29. Ramos TG, Melo ACMA (2006) An extensible resource discovery mechanism for grid computing environments. In: 6th IEEE CCGRID, vol 1, pp 115–122
30. Rong P, Pedram M (2003) Extending the lifetime of a network of battery-powered mobile devices by remote processing: a Markovian decision-based approach. In: Proceedings of DAC '03. ACM Press, New York, pp 906–911
31. Sajjad A, Jameel H, Kalim U, Han SM, Lee YK, Lee S (2005) Automagi—an autonomic middleware for enabling mobile access to grid infrastructure. In: ICAS-ICNS. IEEE Computer Society, Washington p 73
32. Schneider J, Linnert B, Burchard LO (2006) Distributed workflow management for large-scale grid environments. In: SAINT. IEEE Computer Society, Washington, pp 229–235
33. Shi W, Li S, Lin X (2006) Towards merging pervasive computing into grid—lightweight portal, dynamic collaborating and semantic supporting. In: IMSCCS, vol 1. IEEE Computer Society, Washington, pp 560–563
34. Shimosaka H, Hiroyasu T, Miki M (2007) Distributed workflow management system based on publish-subscribe notification for web services. New Gener Comput 25(4):395–408
35. Silva APC, Borges VCM, Dantas MAR (2008) A framework for mobile grid environments based on semantic integration of ontologies and workflow-based applications. INFOCOMP J Comput Sci 7(1):60–69
36. Studer R, Benjamins R, Fensel D (1998) Knowledge engineering: Principles and methods. IEEE Trans Knowl Data Eng 12(25):161–197
37. Sun Java Wireless Toolkit (2008) Java 2 Platform, Micro Edition (J2ME) Wireless Toolkit. Web Page [Online]. Available in: http://java.sun.com/products/sjwtoolkit/. Last access on 12th May 2011
38. Xing W, Dikaiakos MD, Sakellariou R (2006) A core grid ontology for the semantic grid. In: 6th IEEE CCGRID, pp 178–184
39. Yu J, Buyya R (2005) A taxonomy of workflow management systems for grid computing. SIGMOD'05 34(3):44–49
40. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl 1(1):7–18