A Framework for the Flexible Deployment of Scientific Workflows in Grid Environments

Javier Fabra, Sergio Hernández, Pedro Álvarez, Joaquín Ezpeleta
Aragón Institute of Engineering Research (I3A)
Department of Computer Science and Systems Engineering
University of Zaragoza, Spain
Email: {jfabra,shernandez,alvaper,ezpeleta}@unizar.es

***Abstract*—Scientific workflows are generally programmed and configured to be executed by a specific grid-based system. The integration of heterogeneous grid computing platforms in order to build more powerful infrastructures and the flexible deployment and execution of workflows over them are still two open challenges. Solutions based on meta-scheduling have been proposed, but more flexible and decentralized alternatives should be considered. In this paper, an alternative framework based on the use of a tuple-based coordination system and a set of mediation components is proposed. As a use case, the First Provenance Challenge has been implemented using two different workflow technologies executed over the framework, Nets-within-Nets and Taverna, and transparently deployed on two different computing insfrastructures. The proposed framework provides users with scalability and extensibility mechanisms, as well as a complete deployment and scheduling environment suitable for a wide variety of scenarios in the scientific computing area.**

Keywords – middleware for integration; scientific workflow deployment; grid-based systems.

I. INTRODUCTION

Grid computing emerged as a paradigm for the development of computing infrastructures able to share heterogeneous and geographically distributed resources [1]. Due to their computational and networking capabilities, this type of infrastructure has turned into execution environments suitable for scientific workflows. Scientific workflows are a type of workflow characterized for being composed by a large number of activities whose execution requires a high computation intensity and complex data management.

Currently, many efforts are being carried out in the field of scientific computing to execute their experiments taking full advantage of grid technologies. Two important open challenges in this area are the integration of heterogeneous grid computing platforms in order to build more powerful infrastructures and the flexible deployment and execution of workflows over them. Some authors have proposed solutions based on the use of meta-schedulings without considering dynamic behaviours or workloads. However, in order to tackle with the nature of grids, it is required to consider more flexible and decentralized alternatives.

In this paper, a framework able to tackle the previous challenges is proposed. As shown in [2], [3], the use of a broker based on the Linda coordination model [4] and a set of mediators facilitates the flexible integration of heterogeneous grid computing environments, addressing the challenge of creating more powerful infrastructures. These components encapsulate and handle specific features of various computing environments integrated into our framework, being programmers unaware of this heterogeneity. As a result, the tasks that compose a workflow can be executed in a flexible way using different computing environments. Unlike current proposals the framework is not based on the use of a meta-scheduler to perform global scheduling decisions, but each computing environment competes to execute jobs according to the availability of its own grid resources. In order to implement this alternative scheduling model, each one of these computing environments is represented in the broker by a specific mediator able to achieve suitable scheduling decisions. Hybrid computing environments could be easily integrated implementing new mediators. On the other hand, scientific workflows can be programmed independently of the execution environment in which they will be executed. The Net-within-Nets paradigm [5] and the Renew tool [6] have been used for programming this type of workflows. This is also compatible with other existing workflow programming languages. Indeed, Taverna workflows can be programmed using the framework services or translated to our programming language and then executed.

The remainder of the paper is organized as follows. Section II introduces some related work. In Section III, the architecture of the framework is presented. The role of the Linda-based broker, its implementation details and task dispatching mechanisms are described in Section IV. The flexible integration of heterogenous grid middlewares and grid management components with the broker is then detailed in Section V. The features and new capabilities are shown by means of an example that implements the First Provenance Challenge in Section VI. Finally, conclusions are depicted in Section VII.

II. RELATED WORK

A considerable progress has been made in the understanding of the particular nature of scientific workflows and the implementation of grid-based systems for their specification, scheduling, and execution. A detailed survey of existing grid workflow systems is presented in [7], [8]. The comparison of several systems shows relevant differences in the building and

 ISBN: 978-1-61208-216-5

execution of workflows that causes experiments programmed by scientists and engineers to be strongly coupled to the underlying grid-based execution system. This coupling forces grid administrators to perform relevant configuration and integration efforts in most of the scientific workflow deployments. Therefore, some interesting challenges are still open: the ability to program scientific workflows independently of the execution environment, the portability of scientific workflows from one execution environment to another, or the integration of heterogeneous execution environments to create more powerful computation infrastructures, for instance. Consequently, research efforts should concentrate on the definition of new high-level programming constructs independent of specific grid technologies and also on the provision of execution infrastructures able to interface multiple providers. This type of infrastructure should integrate software adaptation layers for translating generic management operations to provider-specific APIs. Additionally, new strategies of resource brokering and scheduling should be integrated into these execution environments to facilitate the utilization of multiple-domain resources and the allocation and binding of workflow activities to them.

Let us briefly resume some of the current proposals for provisioning flexible and extensible execution infrastructures. On the one hand, different grid-based systems built on a *meta-scheduler* have been proposed [9], [10], [11]. A meta-scheduler is a middleware component that provides advanced scheduling capabilities on a grid consisting of different computing platforms. The software architecture of all these solutions is very similar and is composed of the following components: a resource monitoring system to collect information from integrated computing platforms, a meta-scheduler to distribute jobs among grid resources using different scheduling policies [12] and, finally, a set of adaptation components to achieve mediation between middleware components and computing platforms. On the other hand, architectures based on the integration of meta-schedulers have been adapted for taking advantage of Cloud technologies [11], [13], [14]. Resulting computing environments comprise of virtualized services usage-based payment models in order to achieve more efficient and flexible solutions, where the supported functionality will be no longer fixed or locked to underlying infrastructure.

III. An Open Framework for Programming and Executing Scientific Workflows

In short, the main goals of our approach are:

- To execute scientific workflows programmed using a High-level Petri nets formalism or other standard languages widely accepted by the scientific community.
- To simultaneously work with different and heterogeneous grid middlewares or with middlewares implemented using different technologies (e.g., Web services). At this respect, workflow execution engines must be uncoupled from specific grid technologies.
- To allow the addition or removal of resources without previous announcement.
- To support different scheduling strategies and policies in the execution environment. The use of a particular scheduling strategy or policy should depend on the characteristics and requirements of each workflow application.

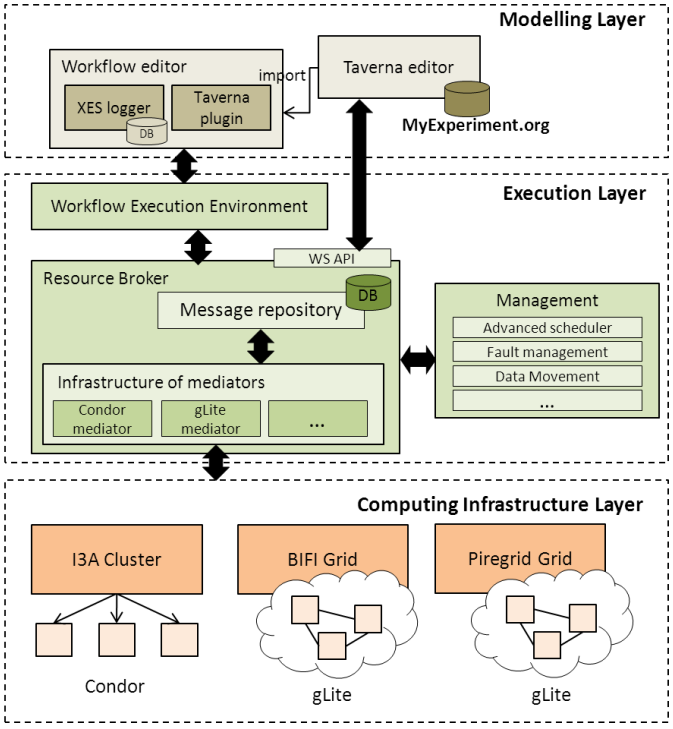

Fig. 1. Architecture of the execution environment.

Figure 1 shows the high-level architecture of the proposed framework. As shown, the architecture consists of three layers: the *modelling layer*, the *execution layer* and the *computing infrastructure layer*. In the following, each layer as well as its main components and interfaces are described in detail.

Firstly, the *modelling layer* consists of a set of tools for the programming of workflow applications. A workflow can be developed using the broker services, which are exposed through its Web service interface, using a workflow modeling tool such as Taverna [15], for instance. Also, we propose the use of Reference nets, a subclass of Petri nets, to implement workflow applications from the perspective of the Nets-within-Nets paradigm [5]. Nevertheless, other high-level programming languages for workflows could be also used by scientific communities (e.g., physicists, biologists or astronomers) for programming their workflows. With respect to this issue, plugins can be added to the modelling layer to support existing or new modelling approaches, such as the Taverna plugin shown in Figure 1, for instance. This plugin allows to import workflows programmed with Taverna, which are automatically translated to the workflow format in the workflow editor and then directly executed. A good repository for these type of workflows is the scientific community hosted at *MyExperiment.org*. In this work, Renew [6] is used as a workflow editor. Renew is an academic open-source tool that

allows the direct execution of Reference nets without any additional coding process and which represents a worth benefit for the final user.

Secondly, the *execution layer* is composed of the core components. The *workflow execution environment* is responsible for controlling the execution of workflows and submitting tasks to the *resource broker* when they must be executed. Internally, the broker consists of a *message repository* and a set of *mediators*. Messages are used to encapsulate any information that is passed through the components of the system. A message can describe a task to be executed or the result of its execution, for instance. Mediators encapsulate the heterogeneity of a specific grid middleware, having a complete knowledge of its capabilities. This knowledge is used for making dispatching decisions (which specific computing infrastructure will execute a pending task?). Subsequently, the grid middleware of the selected computing platform will schedule the set of resources needed for executing the task. As a result, the broker uncouples the workflow execution environment from the specific details about the grid-based computing infrastructures where tasks will be executed. This design avoids the need for a close integration of the workflow execution environment with specific grid middlewares used for the execution of tasks.

Let us now go deeper into the description of the two components of the broker. On the one hand, the Linda coordination model [4] has inspired the implementation of the message repository. Messages are encoded as tuples and stored into a tuple space. The interface of the repository provides a set of operations for accessing the tuples stored in the tuple space according to the semantics of Linda. In Section IV, we will depict the advantages of using a Linda-based repository and provide details about its implementation. On the other hand, *mediators* are required for achieving the aforementioned uncoupled integration. In general, a mediator is an entity that directly communicates with the tuple repository, matches and retrieves special-tagged tuples and processes them. In our approach, each grid middleware is represented by a mediator. Internally, this mediator is responsible for: i) having a complete information of the grid resource it represents; ii) interacting with the tuple repository to find at run-time tasks that could be executed by the set resources of its middleware; iii) dispatching the task to the middleware for its execution and controlling the input and output data transference; and, finally, iv) storing the results of the executed task in the tuple repository as tuples. Mediators of different and heterogeneous grid middlewares could compete for the execution of a specific task. Currently, as it will be described in Section V, different mediators have been implemented for the grid middleware we have access to (Condor and gLite) and then integrated into the *infrastructure of mediators*.

On the other hand, a set of *management components* has also been integrated into the execution layer to support the execution of workflow applications: the fault management component, the data movement component or the advanced scheduling component, for instance. The integration procedure of these components is similar to the one used by mediators. A management component interacts with the tuple repository in order to match and retrieve special-tagged tuples and then processes them. Therefore, the action of these components can be triggered as a result from the previous processing, which allows to dynamically compose complex action chains. In Section V the component for the fault management subsystem and its integration will be detailed.

Finally, the *computing infrastructure layer* is composed of different and heterogeneous computing platforms. The interaction with these platforms is managed by the corresponding grid middlewares. Currently, three computing platforms are integrated in the framework we manage: the HERMES cluster hosted by the Aragón Institute of Engineering Research (I3A), which is managed by the Condor middleware; and the two research and production grids managed by the gLite middleware and hosted by the Institute for Biocomputation and Physics of Complex Systems (BIFI) belonging to the European Grid Initiative (EGI), namely AraGrid and PireGrid.

To sum up, the open nature of the proposed solution is provided by the resource broker, composed of a Linda-based repository and a set of mediators, providing scientists with a high level of abstraction and flexibility when developing workflows. On the one hand, workflow programmers must concentrate on the functional description of workflow tasks and corresponding involved data. Specific details about the computing platforms where these tasks will be executed are ignored from the programmer perspective. On the other hand, the message repository facilitates the integration of mediators and management components and the scalability of the overall framework. Currently, its dispatching model is based on the functional capabilities of the computing platforms managed by the set of mediators. And, finally, these mediators are responsible for encapsulating the technological heterogeneity of the different types of grid middlewares and resource-access technologies (e.g., Web services). New mediators may be easily added in order to integrate new middlewares or technologies.

IV. LINDA-BASED TASK DISPATCHING

As previously stated, the resource broker is composed of a message repository and a set of components (mediators) that interact through this space by means of the exchange of messages. In this section, the role of the Linda-based message repository and the corresponding task description and dispatching mechanisms are presented.

Linda [4] is a coordination model based on two notions: tuples and a tuple-space. A tuple is something like [”Gelernter”, 1989], a list of untyped values. The tuple space is a collection of tuples stored in a shared and global space that can be accessed with certain operations, that allow processes to read and take tuples from and write them into it in a decentralized manner. For instance, the operation `in(x,["Gelernter", ?])` tries to match the *template* `["Gelernter", ?]`, which contains a wildcard, with a tuple in the shared space. If there is a match, a tuple is extracted from the tuple space and

assigned to variable `x`; otherwise, the process blocks until a matching tuple appears. The matching is free for the wildcard, but literal for constant values. The Linda matching mechanism allows easily programming distributed synchronization processes.

Linda-based coordination systems have been widely used for communicating and coordinating distributed processes. Their success in distributed systems is due to a reduced set of basic operations, a data-driven coordination and a space and time uncoupled communication among processes that can cooperate without adapting or announcing themselves [16].

Let us now introduce how tuples are describe and dispatched in our appraoch. Tuples are used to code the information needed for submitting a job to a grid middleware or recovering the result (or an exception) of an executed job. A tuple structure based on the *Job Submission Description Language* standard, JSDL [18], has been adopted. From the job submission point of view, this representation includes the specification of the application to be executed, the references to input and output data (represented by the corresponding URIs), a description of the host required for its execution (operating system, CPU architecture and features, memory, network bandwidth, etc.), QoS parameters and, optionally, the grid middleware responsible for its execution. In case the target grid platform is not specified, different mediators compete for the job execution in base to certain policies. On the other hand, a result tuple contains a reference to the original request, a reference to the output data and the execution log (grid and host used for the job execution, execution costs and QoS results, mainly). If an error occurs, the result tuple will contain the information about it. The fault handling component, which handles these faults, will be depicted in Section V.

Once the tuple representing a job has been created, the workflow execution environment puts it into the message repository by means of an `out` operation. Each grid computing platform is connected to the platform by means of a mediator, which knows the applications that could be locally executed by its grid and the description of the available internal resources. Each mediator is then waiting for tuples that encode such job requests able to be executed by its grid. Obviously, this waiting will depend on the availability at run-time of the grid and its capabilities. An `in` operation is invoked by the mediator in order to retrieve a tuple of its interest, using the Linda matching mechanism. Then, the retrieved tuple is locally processed by the mediator to perform the corresponding invocation to the grid middleware it represents.

If many grid computing platforms are able to execute a job, their mediators will compete to retrieve the job request tuple. The Linda matching mechanism is non-deterministic and, therefore, it does not offer any further guidance about which mediator will retrieve the job request tuple. In this work, the use of WS-PTRLinda, an extension of a previous distributed Linda-based implementation of a message broker, called *DRLinda* [17], is proposed. As DRLinda, WS-PTRLinda was developed using Nets-within-Nets and the Renew tool, the same technologies we used for programming workflow applications. WS-PTRLinda provides a new Web-service based interface (SOAP 1x. SOAP2 and REST), support for persistence of the tuple space (for high-availability demanding environments), and a timeout mechanism useful for failure detection. Currently, a basic and non-deterministic scheduling is being used for dispatching job requests to grid mediators. In [17], we proposed and implemented some alternative matching mechanisms to solve specific problems. Similarly, new grid-oriented matching mechanisms could be defined to extend the scheduling policies of the broker (e.g., a QoS-based scheduling policy). Let us finally comment on two relevant advantages of this Linda-based brokering. Firstly, the cooperation is uncoupled because the execution environment does not have any prior knowledge about mediators and vice versa. The interaction style is adequate enough to be used in environments where it is very important to reduce as much as possible the shared knowledge between different components. Also, writing and reading components can cooperate without adapting or announcing themselves. New mediators could be added/removed without affecting the rest of components integrated into the framework.

V. FLEXIBLE INTEGRATION OF GRID MIDDLEWARES

Following the presented approach, different types of resources and components (execution engines, management components or mediators, for instance) can be integrated in an easy and uncoupled way. The only requirement for these components is to implement the Linda coordination API in order to put and remove tuples. Besides, components can be added or removed dynamically and transparently to the rest of the system, facilitating this way the scalability and adaptation of the framework.

In this section, two different types of integrated components are presented. The first one is a mediator able to interact with the Condor middleware, whereas the second one is a fault management component. When a fault is detected during the execution of a job, this component will re-schedule the job according to different policies. Our aim is to illustrate how this solution is able to interact with grid computing platforms managed by heterogeneous grid middlewares.

A. Interaction with the Condor middleware

As previously described, the framework is able to interact with several underlying grid infrastructures. Let us depict how a mediator has been developed to integrate a Condor middleware. Specifically, this mediator is responsible for the interaction with the HERMES cluster. Figure 2 shows the functional components of the mediator required for supporting such interaction. Additionally, this mediator can be reused for interacting with any computing platform managed by Condor.

The *Job Manager* interacts with the Linda-based broker depicted in the previous section in order to read job requests and write their results. Obviously, all request types that could be fulfilled by the cluster must be known by the manager. For this purpose, the *Internal Resource Registry* knows the list of applications that could be locally executed and the

 ISBN: 978-1-61208-216-5

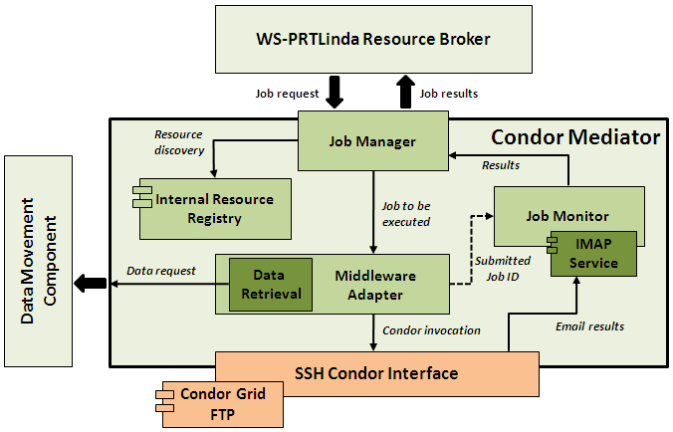

Fig. 2. Components of the Condor mediator.

description of available internal resources. This registry should monitor the cluster and dynamically update its information, but at this first implementation of the Condor mediator this information is static. Once a job request has been retrieved, the manager sends it to the *Middleware Adapter* component that is responsible for translating the request into a Condor job. Before submitting the job to the cluster via the SSH protocol, the adapter internally carries out two important tasks. First, it assigns an identifier to the job (*Job ID*) and sends it to the *Job Monitor* component. This ID will be used to correlate jobs and tuples. In case the input data required by a job are stored in an external computing platform, the adapter interacts with the Data Movement component for moving them (or making a copy) into the Condor cluster. After that, the adapter submits the job to the Condor middleware.

Internally, Condor can schedule the execution of submitted jobs depending on the local state of its resources. The goal is to achieve the best possible throughput. Therefore, a double scheduling can be done in the approach, similarly to the hierarchical scheduling model described in [19]. Once the job execution has been completed, results are sent through a logging mechanism (in our case, SMTP-IMAP) service integrated in the *Job Monitor*. This component maps received results with job requests and forwards them to the job manager. Finally, results are written in the broker so they can be then taken by the workflow application that submitted the original request.

This design and implementation is quite flexible and provides reusability. For instance, we have also developed a mediator to interact with the gLite middleware used in AraGrid. Its design is similar to the previous one. In fact, most internal components have been reused, as the job manager and the internal resource registry, and others components have been adapted, as the middleware adapter or the job monitor, for instance.

B. Fault handling

When dealing with scientific workflows, failures can arise at several levels. In this work, we will focus on those faults and exceptions that happen at the execution level. When the execution of a job fails, the corresponding mediator captures the fault and puts an error tuple into the message repository. This tuple, which will be processed by the *Fault management component*, contains information about the cause of the fault that will be used by the manager to take a decision with respect to the job execution. Different decisions could be taken: to submit the job again to the same grid computing platform, to submit the job to an alternative and reliable grid computing platform or to notify the error to the workflow executing environment in case the error persists, for instance. In the last case, most grid solutions offer two different ways to manage the fault: corrective actions or alternative workflows.

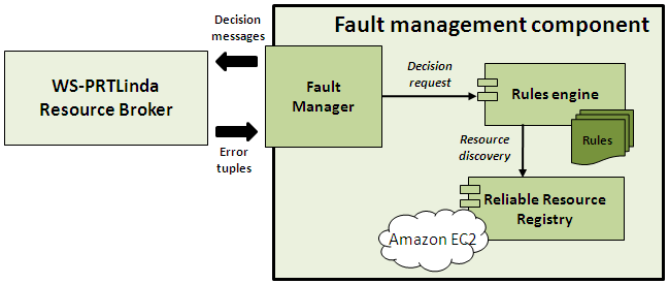

Fig. 3. Components of the fault management component.

Figure 3 shows the internal design of the fault management component. A *Fault Manager* interacts with the message repository in order to retrieve error tuples and to write the corresponding decision tuple. When an error tuple is found, the fault manager processes it and creates a decision request that is sent to a decision maker. We have used a *rules engine* as the decision maker. Rules are encoded in RuleML (the standard Web language for rules using XML markup [20]) and describe the corrective actions that will be executed in case of each type of error. These actions can be changed and modified at runtime, providing adaptation capabilities based on specific scenarios. Normally, the job will be sent again for a new execution on the corresponding infrastructure. However, in case it fails again or even if the error tuple contains some critical information, a usual action is to send the job request to a reliable grid middleware (our ultimate goal is the successful execution of job requests). Reliable grid middlewares have special characteristics (number of nodes, throughput, rejection rate, etc.), which turn them into more suitable candidates for a difficult job execution. For this purpose, a *Reliable Resource Registry* has been implemented and integrated in the fault management component. The current version of the registry contains a list of reliable grid middlewares. This list is used by the rules engine to decide in which middleware the failed job request will be executed. Finally, the fault manager puts a new job request tuple into the broker, specifying the grid middleware responsible for its execution.

VI. A CASE STUDY: THE FIRST PROVENANCE CHALLENGE

As a case study we present a workflow implementing the First Provenance Challenge [21].

 ISBN: 978-1-61208-216-5

The goal of the First Provenance Challenge (FPC) workflow is to create a *brain atlas* from an archive of four high resolution anatomical data and a reference image. Some image processing services are required for the workflow execution. These services have been deployed into heterogenous grid middlewares (more specifically, into the Condor cluster hosted by the I3A Institute and the gLite grids hosted by the BIFI Institute). In this example we show the flexibility of our proposal: some jobs are programmed to be executed by a specific computing platform, and other jobs may be executed by any available computing platform able to invoke the required service.

The workflow requires seven input parameters, whose specific values are implemented as the initial markings of places `Grid_Environment`, `Reference_image`, `Input_image_{1..4}`, and `Images_directory`. Their meanings are, respectively: the URL of one of the clusters where the workflow is going to be executed (more specifically, the cluster hosted by the I3A), the URI of the reference image, the URIs of the four images to be processed and the directory where the intermediate and final image files will be stored.

Figure 4 shows the implementation of the workflow using the Renew tool. Due to space limitations, only the first image processing flow is detailed in the figure, although the remaining branches for anatomy `Image2`, `Image3` and `Image4` are similar. Alternatively, Figure 5 depicts the implementation of the same workflow using Taverna. Job requests and results are encoded as Linda tuples. A request tuple is a nested tuple composed of four elements: the application or service to be executed and the URIs of the input and output data, the file descriptors for standard streams, QoS parameters and the computing platform where the request is going to be executed, respectively. Let us explain a tuple example, specifically the tuple depicted in transition `Align_warp_1(out)`. By putting that tuple in the message repository, the `Align_warp` service is invoked by the corresponding mediator using as input data an anatomy image, a reference image and their headers. The output is a warped image. For the sake of simplicity, file descriptors and QoS parameters are omitted in the tuple. Finally, the initial marking of the `grid_environment` place determines the value of the `grid` variable and, therefore, the computing platform selected for the job execution (the first field of this last tuple contains the access information required by the platform).

Tuples are either built and put into the message repository by means of the `Broker.out` action (as in the `Align_warp_1 (out) transition`, for instance) or withdrawn from the broker by means of the `Broker.in` action (as in the `Align_warp_1 (in) transition`, for instance). The sequential execution of these couple of transitions for a given image corresponds to an asynchronous call to the `Align_warp` service: first, the tuple with the information is put into the message broker, then the corresponding mediator takes it and invokes the service, putting the invocation result into the broker as a tuple and finally the result is captured and put into the workflow net by means of

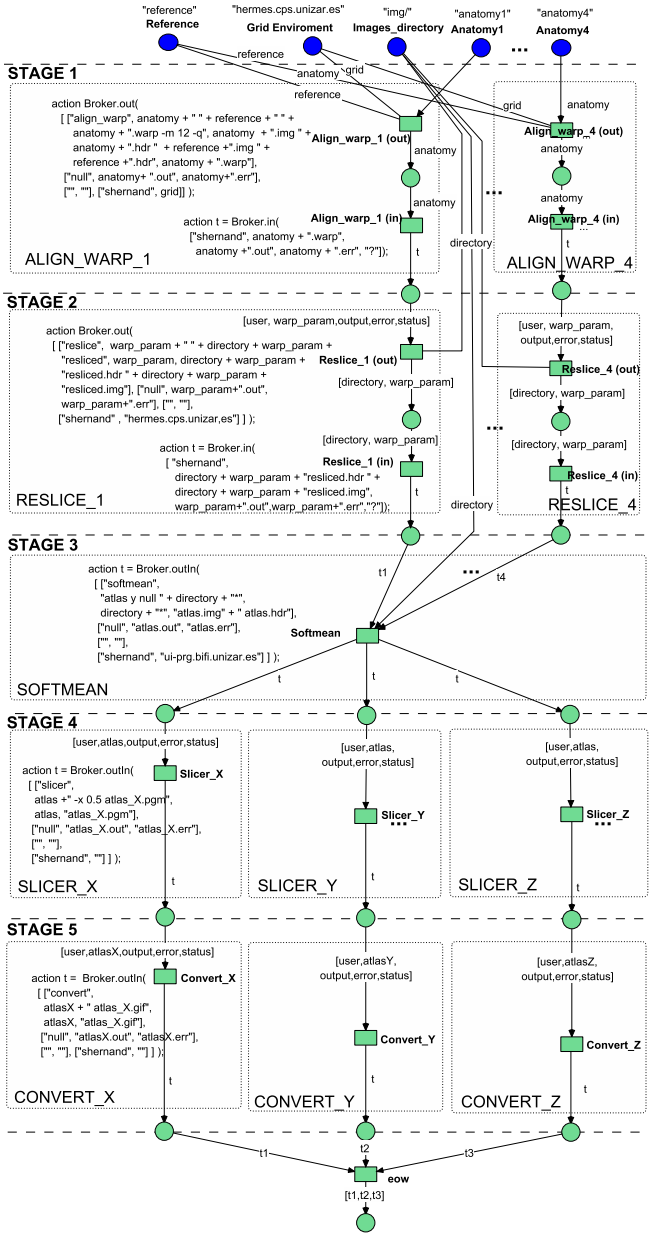

Fig. 4. Nets-within-Nets based implementation of the First Provenance Challenge workflow.

the second transition. Given the semantics of Petri nets, the processing of the input images can be done in any interleaved way, since tuples are put/removed into/from the broker as soon as resources are available. In this first stage the job request is executed in the cluster specified by the initial marking (the `grid` variable is an input parameter of the request submitted to the broker by the `Align_warp (out)` transition).

Once stages 1 and 2 are finished, Stage 3 takes the whole set of images from the directory specified by the parameter `Images_directory`, and executes the `softmean` method with these images as an input. At this stage the service deployed in one of the grids hosted by the BIFI institute is explicitly invoked. The last job request and its result are

 ISBN: 978-1-61208-216-5

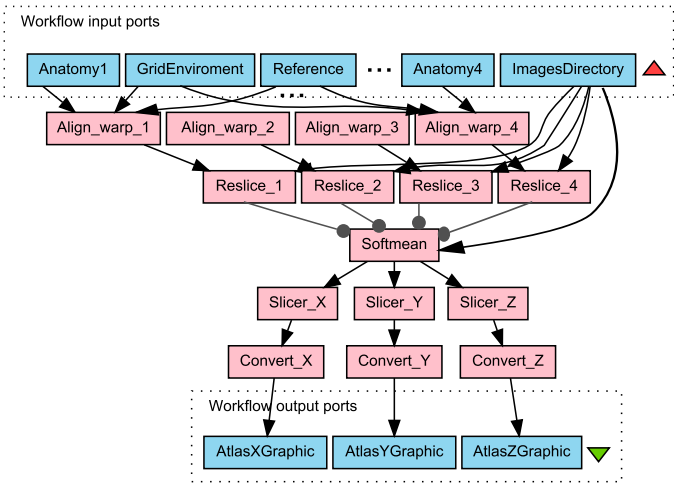

Fig. 5. Taverna implementation of the First Provenance Challenge workflow.

carried out by means of the `Broker.outIn` action: from the workflow point of view this corresponds to a synchronous call to the service described in the tuple. Then, softmean results are distributed so that stages 4 and 5 could be executed in parallel to compute the atlas data set for each dimension in axis x, y and z. The slicer and convert jobs could be executed by any available computing platform. Therefore, different executions of the workflow could invoke services deployed in different platforms. Finally, firing of transition `eow` (*end-of-workflow*) terminates the workflow. The resulting images will have been stored in the images directory.

Figure 5 depicts the workflow implemented with Taverna (some flow symbols in the top of the figure have been removed to improve readability). As shown, the structure is similar to the Nets-within-Nets implementation, although in this case the workflow is composed of several subworkflows, each of them implementing the previous invocations to the broker in order to put and withdraw tuples. Due to space limitations, the description of these subworkflows is left out of this paper.

A. Flexible deployment and execution

In order to analyze and test the transparency and flexibility of the proposed approach, the First Provenance Challenge workflow was executed using the framework. The target computing infrastructure for the execution of each stage (which can be specified in out transitions at each stage in Figure 5) was left unset, meaning that the mediators compete for each submitted task. At this respect, both HERMES and AraGrid were setup to separately allow the execution of the FPC workflow. However, as the aim of this experiment was to improve the overall execution cost of the workflow, the advanced scheduling component was programmed to perform a meta-scheduling process considering the load of the underlying computing infrastructures and the history of previous executions. Therefore, at every moment the best suitable candidate is estimated, avoiding the dispatching of a task to an overloaded infrastructure. This means that each task is first captured by the advanced scheduling component and then the target infrastructure is set (so, the corresponding mediator will retrieve the task for its execution). However, the whole process is transparent from the user's perspective.

To do that, the advanced scheduler also considered the average load of each infrastructure at every moment. Figure 6 depicts the daily average load (% of the maximum load) in the HERMES and AraGrid computing infrastructures. As it can be observed, both computing infrastructures have different load models. Their trends during the day as well as the previous execution time are used to decide the most suitable candidate for each task deployment.

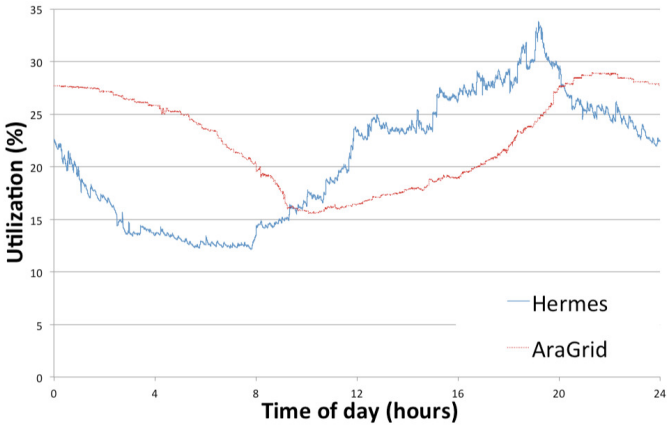

Fig. 6. Hermes and AraGrid daily utilization (in percentaje).

Figure 7 depicts the results obtained for 900 executions of the FPC workflow deployed on the framework. Average execution times (in seconds) are shown for each separated infrastructure (HERMES and AraGrid) and also for the framework for each stage of the First Provenance Challenge workflow. The overall execution time (average) is better when using the framework. This is due to the best candidate selection performed by the advanced scheduler (in most cases). The analysis of each separated stage depics that most of the time (70%) the HERMES cluster computing infrastructure gets a better execution time that AraGrid, which is related to the fact that the framework execution time is closed to the HERMES one.

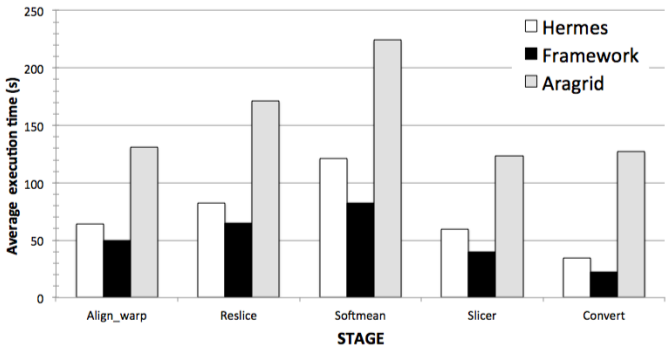

Fig. 7. Experimental results for the First Provenance Challenge workflow.

If we consider the average execution times for the complete workflow, AraGrid got the worst results with 777 seconds,

HERMES got 362 seconds and the framework got 260 seconds. Obviously, using the most adequate infrastructure to get the better execution time is not a trivial process from the researcher's point of view. However, by means of the use of the framework, this is done in a flexible and transparent way. Other possibilities are to reduce access costs (for instance, if each computing hour has an asssociated cost), resource usage, etc. Regarding the time to move data between the two infrastructures (as output from a stage is used as input of the following one), the average time for each workflow execution was less than 55 seconds (so the average framework execution time goes to 315 seconds).

VII. Conclusion and future work

In this paper, a framework to solve some of the open challenges in the application of grid-based solutions to scientific workflows has been presented. This framework is uncoupled from specific grid technologies, able to work simultaneously and transparently with different and heterogeneous grid middlewares, providing scientists with a high level of abstraction when developing their workflows. The integration of the execution environment with different grid middlewares has been carried out by means of a resource broker composed of a Linda-based coordination system and a set of mediators. Thanks to the aforementioned broker, this integration is flexible and scalable. On the other hand, regarding the workflow programming point of view, the proposal is also open and flexible. As it has been shown, workflows programmed using standard languages or existing service-oriented workflow management systems (e.g., Taverna) can also be executed in the framework.

Currently, the proposed framework is being applied to solve some complex and high time-consuming problems, such as the behavioural analysis of semantically-annotated scientific workflows, or the analysis of existing data connections into the Linked data cloud, for instance. These solutions will allow improving the capabilities of the presented approach and also analyzing its performances.

We are also working on the integration of Cloud-related solutions, such as using the *Amazon Elastic Cloud Computing Simple Queue Service* (Amazon EC2 SQS) in order to have an alternative message repository, as well as providing specific high-performance computing capabilities (indeed, currently Amazon EC2 offers a mechanism to virtualize a HPC machine, able to handle critical and complex computation tasks). Related to this last point, we are adding some external reliable computing platforms by means of virtualization technologies. In [2] we sketched the implementation of a similar mediator able to support the execution of business tasks. Similarly, a new mediator able to submit job requests to the EC2 interface with the required policies has been implemented.

Acknowledgment

This work has been supported by the research project TIN2010-17905, granted by the Spanish Ministry of Science and Innovation.

References

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Second edition, Morgan Kaufmann Publishers, 2004.

[2] J. Fabra, P. Álvarez, J.A. Bañares, and J. Ezpeleta, *DENEB: A Platform for the Development and Execution of Interoperable Dynamic Web Processes*. Concurrency and Computation: Practice and Experience, Vol. 23, Issue 18, pp. 2421-2451, 2011.

[3] R. Tolosana-Calasanz, J.A. Bañares, P. Álvarez, and J. Ezpeleta. *Vega: A Service-Oriented Grid Workflow Management System*. In 2nd International Conference on Grid computing, High Performance and Distributed Aplications (GADA'07), vol. 4805, pp. 1516-1523, 2007.

[4] N. Carriero and D. Gelernter, *Linda in context*. Communications of the ACM, Vol. 32, Num. 4, pp. 444-458, 1989.

[5] O. Kummer, *Introduction to petri nets and reference nets*. Sozionik Aktuell, Num. 1, pp. 19, 2001.

[6] O. Kummer and F. Wienberg, *Renew - the Reference Net Workshop. Tool Demonstrations*. In 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), pp. 87-89, 2000.

[7] M. Rahman, R. Ranjan, and R. Buyya, *A Taxonomy of Autonomic Application Management in Grids*. In 16th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010), 2010.

[8] J. Yu and R. Buyya, *A taxonomy of workflow management systems for Grid Computing*. Journal of Grid Computing, Vol. 3, Issues 3-4, pp. 171-200, 2005.

[9] E. Huedo, R.S. Montero, and I.M. Llorente, *A Framework for Adaptive Execution on Grids*. Software - Practice and Experience , Vol. 34, Issue 7, pp. 631-651, 2004.

[10] M. Heidt, T. Dornemann, J. Dornemann, and B. Freisleben, *Omnivore: Integration of Grid meta-scheduling and peer-to-peer technologies*. In 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), pp. 316-323, 2008.

[11] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, and I.M. Llorente, *Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers*. Future Generation Computer Systems, Vol. 28, pp. 358-367, 2012.

[12] J. Yu and R. Buyya, *A novel architecture for realizing Grid Workflows using Tuple Spaces*. In 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004), Pittsburgh, USA, pp. 119-128, 2004.

[13] G. Mateescu, W. Gentzsch, and C.J. Ribbens, *Hybrid computing: Where HPC meets grid and Cloud computing*. Future Generation Computer Systems, Vol. 27, pp. 440-453, 2011.

[14] Y. Zhang, C. Koelbel, and K. Cooper, *Hybrid re-scheduling mechanisms for workflow applications on multi-cluster grid*. In 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), pp. 116-123, Shanghai (China), 2009.

[15] *Taverna: an open source and domain independent Workflow Management System*. Available at http://www.taverna.org.uk/ [retrieved: May, 2012]

[16] P. Álvarez, J.A. Bañares, and P.R. Muro-Medrano, *An Architectural Pattern to Extend the Interaction Model between Web-Services: The Location-Based Service Context*. In 1st International Conference on Service Oriented Computing (ICSOC 2003), Lecture Notes in Computer Science, Vol. 2910, pp. 271-286, 2003.

[17] J. Fabra, P. Álvarez, and J. Ezpeleta, *DRLinda: A distributed message broker for collaborative interactions among business processes*. In 8th International Conference on Electronic Commerce and Web Technologies (EC-Web 2007), Lecture Notes in Computer Science, Vol. 4655, pp. 212-221, 2007.

[18] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, *Job Submission Description Language (JSDL) Specification, Version 1.0*. Available at http://www.gridforum.org/documents/GFD.56.pdf [retrieved: May, 2012]

[19] R. Sharma, V.K. Soni, M.K. Mishra, and P. Bhuyan, *A survey of job scheduling and resource management in grid computing*. World Academy of Science, Engineering and Technology, Issue 64, pp. 461-466, 2010.

[20] H. Boley, *Rule Markup Language, RuleML Specification. Version 1.0.*. Available at http://ruleml.org/ [retrieved: May, 2012]

[21] L. Moreau, B. Ludscher, I. Altintas , R.S. Barga, S. Bowers, and S. Callahan, *The First Provenance Challenge*. Concurrency and Computation: Practice and Experience, Vol. 20, Issue 5, pp. 409-418, 2008.

 ISBN: 978-1-61208-216-5