A One-Stop, Fire-and-(Almost)Forget, Dropping-Off and Rendezvous Point*

R. Menday[1], B. Hagemeier[1], B. Schuller[1], D. Snelling[2], S. van den Berghe[2], C. Cacciari[3], and M. Melato[4]

[1] Central Institute for Applied Mathematics, Forschungszentrum Jülich, D-52425 Jülich, Germany
[2] Fujitsu Laboratories of Europe Ltd, Hayes Park Central, Hayes End Road, Hayes, Middlesex, UB4 8FE, UK
[3] CINECA, via Magnanelli 6/3, 40033 Casalecchio di Reno, Bologna, Italy
[4] NICE, via Marchesi di Roero 1, 14020 Cortanze, Italy
r.menday@fz-juelich.de

Abstract. In order to foster uptake by scientific and business users we need an easy way to access Grid resources. This is the motivation for the A-WARE project. We build upon a fabric layer of Grid and other resources, by providing a higher-layer service for managing the interaction with these resources - A One-Stop, Fire-and-(almost)Forget, Dropping-off and Rendezvous Point. Work assignments can be formulated using domain specific dialects, allowing users to express themselves in their domain of expertise. Both Web service and REST bindings are provided, as well as allowing the component to be embedded into other presentation technologies (such as portals). In addition common desktop notification mechanisms such as Email, RSS/Atom feeds and instant messaging keep users informed and in control. We propose using the Java Business Integration specification as the framework for building such a higher-level component, delivering unprecedented opportunities for the integration of Grid technologies with the enterprise computing infrastructures commonly found in businesses.

1 Introduction

UNICORE[23],[18] has gained a reputation as a vertically integrated architecture. Sometimes referred to as a 'stovepipe' architecture, it offers a complete 'ready to run' solution. From a user and administrative perspective this is clearly attractive.

Recently, in the Grip[11] and then the UniGrids[9] projects, UNICORE has been prominent in promoting interoperable Grid middleware. Indeed, UNICORE emerged as an early adopter of Service Oriented (SOA)[20] approaches to building distributed systems[21]. The consequence of a good SOA design is that there is a loose-coupling between the components, thus loosening the links in the UNICORE stovepipe. Emerging from the current activity in the UNICORE

* This work is partially funded through the European A-WARE project under grant FP6-2005-IST-034545.

W. Lehner et al. (Eds.): Euro-Par 2006 Workshops, LNCS 4375, pp. 225–234, 2007.

community will be a best-of-breed packaging of select components. In essence, the next-generation of UNICORE will become a stovepipe construction toolkit. Furthermore, there now exists the possibility for others to take individual components and use them for something else.

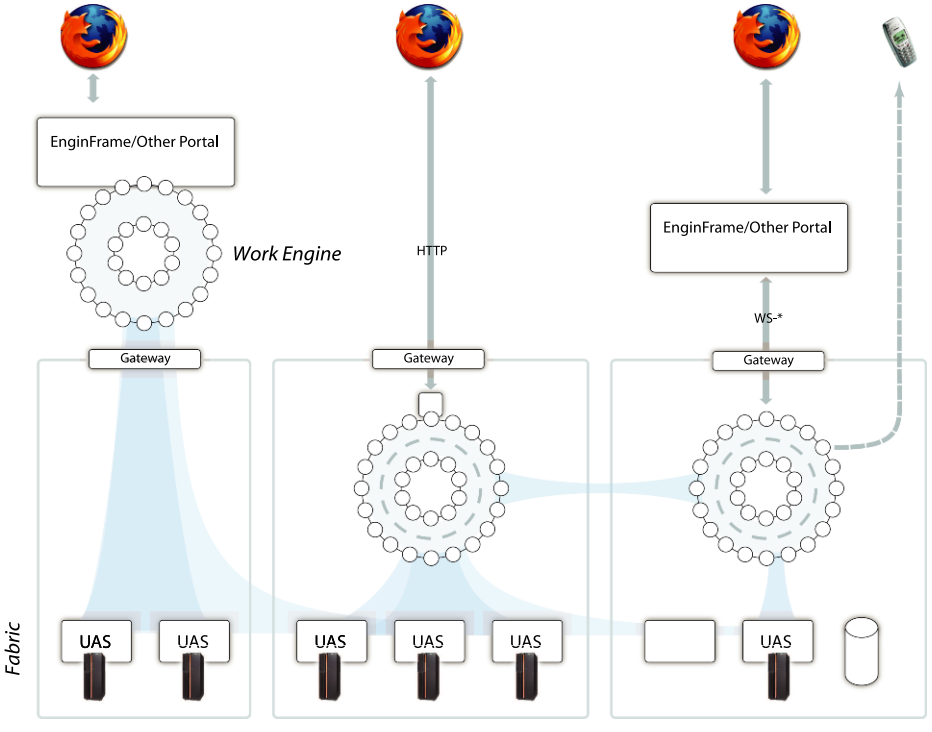

Fig. 1. Deployment possibilities of the work engine

Referring to Figure 1 we assume a cloud of services consisting at the lowest level of fabric services. These are services associated with particular computational or data resources. The UNICORE Atomic Services (UAS) developed in the UniGrids project provide us with a Web services based interface to such atomic grid functionality. Services which are not fabric services - i.e. not coupled to a particular computational or data resource - are termed higher-level services. Individual fabric services are not normally used in isolation. A set of resources and services are orchestrated into a complex workflow, business process, or service chain. This paper deals with such a 'work engine', acting on behalf of multifarious users that manages the multiple invocations of contributing services. This work will be carried out within the A-WARE project[1]. Examples of such functionality include atomic Grid jobs, other higher-level services, databases, legacy applications, etc. In short our work engine component can be described as a

one-stop, fire-and-(almost)forget, dropping-off and rendezvous point.

One-stop, because it presents a façade of the Grid to the user[1]. Fire-and-forget implies that the work engine manages the orchestration of the users' tasks over the Grid infrastructure. In many cases, after assigning work to the engine, the next contact the client makes is when the assignment is complete - the rendezvous. The 'almost' proviso implies that the client may wish to be notified during the execution of the assignment, either for purely informational purposes, or to participate in its execution, approving resource selection choices, or adding additional information not available at the time of submission.

Whilst the use-cases driving the development of the Grid lead to some special requirements, it has also become increasingly evident that businesses face similar issues related to internet-scale communication, cross-organisational interactions, and the accessing of services over the internet. This has resulted in the blurring of the lines between Grid computing in the scientific community, and the kind of the enterprise computing seen within business. Through the seamless integration of Grid resources and local (non-Grid) resources, using a very powerful and flexible orchestration environment, leading to the *disappearing grid*.

The further integration of a Grid portal component, such as EnginFrame[8], would allow organizations to provide application oriented computing and data services to users in a simplified Web browsing experience. Grid portal technology hides the complexity of the underlying Grid infrastructure and provides an additional user-oriented abstraction layer on the Grid.

This paper proposes using the Java Business Integration (JBI)[14] specification for building higher-level services for the Grid. We suggest that this will increase the ease of integration of Grid technologies into standard procedures and systems, bringing considerable advantage through extending the reach of Grid technologies. JBI provides the environment for the orchestration of resources. The JBI based work engine supports a wide variety of work description documents, external bindings, swappable and co-existing orchestration strategies

Domain Specific Languages (DSL) are used throughout the architecture. These can be used to expose a legacy application or process, or to provide core support for pre-designed 'canned' workflows which can be used as a top-level work description, or as embedded fragment in a larger orchestration language. As such, the use of DSLs can be seen as a logical progression of the software resource concept of UNICORE.

The design encourages an ecosystem of multiple clients all using the services offered by the work engine. These are supported through multiple external bindings. So, for example, it will offer a Web services interface as well as HTTP interface following the REST [19] architectural guidelines. Alternatively, the work engine can be deployed as an embedded component in other publishing frameworks, supporting the established portal technology EnginFrame[8], as well as opensource portlet[17] containers such as GridSphere[12]. Finally, there are further opportunities for building domain specific workbench applications leveraging the DSL support.

[1] For performance reasons data transfers occur in a point-to-point nature (bypassing the work engine).

This paper begins in section 2 by reviewing the status of UNICORE development highlighting the UNICORE Atomic Services (UAS) developed in the UniGrids project. We follow with high-level view of functionality targets in section 3. The JBI-based framework is introduced in section 4. In section 5 we outline some future strategies for workflow execution, including the use of rule technologies in section 5.3. Furthermore, we outline in section 5.1 how Domain Specific Languages are a core concept in the architecture. Finally, we conclude with a summary.

2 Atomic Services and Interoperation

Our primary interface for Grid tasks is the UNICORE Atomic Services (UAS) as developed in the UniGrids [9] project. The UAS covers the basic use-cases for 'atomic' Grid usage, e.g. submit and manage a job, elementary data management, at a single target system (a VSite in UNICORE terminology). This is done by defining a contract for Target System and related services.

At the time of writing, nothing exists as a standard - from the GGF (or elsewhere) - with the same level of usability and maturity as the UAS, although the Global Grid Forum has a number of initiatives in this area. Thus, for now, we support the UAS interface as the 'native' interface to atomic Grid functionality, until a concrete standard emerges. Indeed the UAS has provided an excellent input to the standardisation process

We introduce the term *willingness* to categorise levels of support. We see gradients of willingness. For example, a fabric service may use JSDL[10] for describing jobs, although alternative mechanisms for conveying this message are possible. Often a partial willingness to comply is due to the very nature of the standard. For example, JSDL has an extensible nature whereby open-content can appear at some points within the document.

What emerges is that some form of mediation strategy is necessary in almost every case. Sometimes this involves some simple protocol translation steps, but in other cases it may mean using ontological techniques to cope with different information models.

3 Functionality

This section contains an incomplete presentation of possible fields where the higher-level service discussed here may prove useful.

- **Workflowing**
  High-level, abstract workflows described by DSLs broken down into low-level, concrete workflows for execution by fabric services. Basic orchestration of concrete workflows.
- **Scheduling**
  Different approaches to scheduling can be enumerated as static, dynamic and hybrid scheduling. Static tasks are completely predefined or directly

authorised by the client. Dynamic tasks consist of a description of work to be done with no particular resources assigned to them. Dynamic scheduling involves lookup of appropriate resources with respect to an associated requirements description. We can combine both approaches in hybrid scheduling strategies.

- **Brokering**
  Grids are subject to constant change. A dynamic broker supports the selection of resources according to the user's policies and currently offered resources. Reaction on changes of resources during execution of workflows closely links brokers and schedulers.
- **Negotiating**
  Most real-life Grid applications involving multiple resources require scheduling and reservation steps. Coordinated time-dependant synchronised starting provides co-scheduling support.
- **Integrating**
  With the Grid mainstream clearly moving towards web services based technologies, solutions supporting clean integration of 'legacy' business systems or processes are necessary.
- **Mediating**
  In an environment dominated by open, extensible messaging formats, often collaboration between services entails some form of mediation. For example, various dialects of WS-Addressing or JSDL might coexist in a Grid.
- **Transforming**
  In a similar vein, data might need transformation steps between services, for example in a multi-step workflow with data transfers.
- **Managing**
  Specific services might expose administrative interfaces, for example security services might offer the possibility to add users or modify user permissions.
- **Informing**
  The massive amount of both static and dynamic information available in next-generation Grids needs to be filtered for various needs - both end user and software agents. Web users are accustomed to using a wide variety of communication tools, such as e-mail, RSS feeds, SMS or instant messaging. These can be profitably leveraged for Grid users. A common use case is notifying users about an interesting status change of some resource, for example when jobs have finished, or results are available.
- **Interacting**
  A particular work assignment may require input from the user during the course of its execution. Such interaction could be used to approve a dynamic resource selection, or could be used to adjust the rules governing the execution.
- **Securing**
  In heterogeneous, truly service oriented Grids, the ability to use and mediate between various trust and security approaches may well become vital. Our work engine will use appropriate security services to achieve this.

4 Java Business Integration

In order to cover the wide-range of possibilities covered by the functional requirements, we selected the Java Business Integration[14] specification as a framework technology. JBI promotes the idea of a loosely-coupled collection of components interacting with each other via the bus. It is an event-driven, component architecture. The specification defines a standard means to assemble integration components which are plugged into a JBI environment and can provide or consume services through it, in a loosely-coupled way. The JBI environment routes the exchanges between these components and offers a set of technical services.

JBI distils SOA concepts into the design of the internal interfaces collected around the bus. As such, JBI encourages the programmer to design and code in a loosely-coupled manner - e.g. between each module of code contributing to the system, there is a cleanly defined contract for the interactions.

JBI offers a lot of potential integration possibilities into existing enterprise Java deployments. Many businesses will find this a particularly compelling aspect of JBI. Furthermore, as a standard Java specification there exists a number of JBI implementations, and lots of opportunity to re-use existing components. A deployment is declaratively configurable and manageable using standard mechanisms. It is easy to 'customise' a particular JBI deployment, for example to support local processes using a DSL (see section 5.1).

Furthermore, JBI is an excellent framework for supporting multiple protocols and transports, through various binding components, such as

- **REST**
  A well designed HTTP interface following the guidelines of the REST architectural style, offers an extremely attractive interface with an extremely low barrier to entry. Through interaction with a REST interface, browsers can construct Web applications using client-side scripting and using AJAX[2] approaches. It is clear that a number of other interesting Web techniques can also be applied here too.
- **WS-***
  Tool support for Web services is excellent. A good toolkit automates a substantial amount of the process of building client tooling for web services. Businesses with commitment and expertise with Web services will find this channel for interaction appealing.
- **Embedded**
  This allows the work engine to be embedded into portals, and other presentation layer technologies.

The goal of each of the binding components above is to ultimately deliver a work assignment to the JBI bus for execution. The user submitting this can configure their work engine with notification preferences, such that they are contactable during and after the execution of their work. We propose using ServiceMix[3] as the implementation of JBI, and this comes with a number of notification mechanisms (such as Email, Jabber[13] messaging, RSS/Atom feeds) 'out of the box'.

5 Orchestration

The JBI-based core is a suitably powerful and generic framework to host the execution of scientific and business processes. Based on interacting with, and learning from, the computing world surrounding it, semi-autonomous agents form the conduit between the bus and the external world, through monitoring of the outside world, and reacting by sending events onto the bus. For example, agents could check and arrange QoS guarantees. Further agents could be responsible for negotiating trust relationships including security token exchange via a security token service. Co-allocated, time-dependent, synchronised start is also possible given an appropriate co-allocation agent.

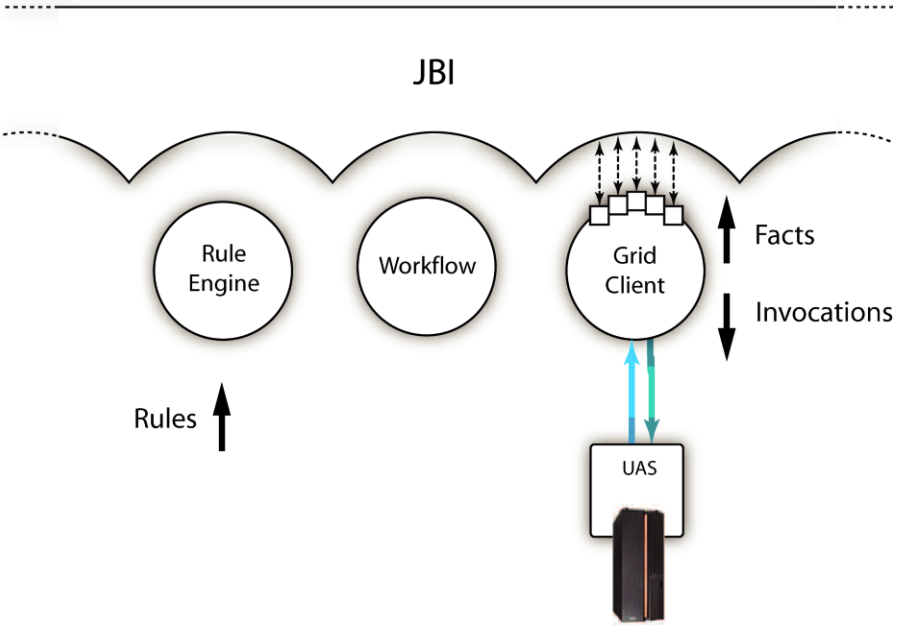

Fig. 2. The bus and some sample components

5.1 Domain Specific Languages

When dealing with workflows on the Grid, one inevitably has to deal with bridging the gap between the low-level, technical workflow (execution of small, "atomic" activities) and the high-level view taken by the end users, where maximising the impact application developers can make in their domain of expertise is important. One approach towards bridging the above-mentioned gap is the combination of orchestration engines and high-level DSLs. The idea is that higher level 'work assignments' use the underlying orchestration services to

execute. Work assignments are, if necessary, mapped to a description understood by the core orchestration components. The work engine can advertise which DSLs it supports.

DSLs allow work assignments to be expressed in the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL descriptions. There is some debate whether XML documents are in fact DSLs[22]. We argue that they are, and that it is the level of abstraction which allows classification as a DSL, although clearly it is nothing more than some XML conforming to a particular schema! We propose supporting both XML and non-XML DSLs as work assignments to be fed into the work engine. However, there is some merit to XML based DSLs[2] as it is then easier to embed inside other XML documents. It is also simple to use XML schema to validate XML-based DSLs.

Regardless of the means by which it was conveyed, at some point a work assignment exists as a message on the JBI bus. Some messages are said to be in a 'native' format, i.e. can be directly executed by one of the orchestration engines which are responsible for managing the workflow and persisting the state of the orchestration. Therefore, a process of transformation and mediation interprets the incoming work assignment. The JBI bus is the controlled environment responsible for managing this break-down. Intermediate steps may themselves be expressed in some DSL formulation. Eventually this process produces an execution plan that can be directly executed in the native format of the underlying orchestration engine.

5.2 BPEL

The Grid community seems to be somewhat split regarding the use of BPEL in conjunction with WSRF. Part of the problem is that the WSRF interactions between a service consumer and provider are quite verbose and fine-grained. While it is possible to describe this conversation as a BPEL workflow the result is somewhat long-winded.

The key is to use the bus for the invocation of services from BPEL. Each invocation breaks down to a series of WSRF-based invocations, but crucially the contract to the BPEL consumer on the bus is coarse-grained and service-oriented, and avoids the verboseness. Thus the usage of JBI as a mediating technology between BPEL and WSRF-based Grid services looks very promising to successfully use BPEL to orchestrate Grid (WSRF-based) services.

Furthermore through JBI multiple orchestration strategies can co-exist. Indeed, runtime selection of orchestration strategy may be based on the type of assignment passed to it. Other orchestration technologies which also look interesting include Business Process Management (BPM) workflow solutions (OSWorkflow[16], jBPM[15], etc), petri-net based solutions (Bossa [4]), continuation-based approaches (bpmscript[5], dalma[6]). As emphasised previously, these orchestration strategies can be swapped in and out much easier using the JBI infrastructure.

[2] Even if its just a trivial wrapping.

5.3 Rules

A rule engine is an example of another useful component that can be hosted by JBI. Prototype work to date has concentrated on the Java rules engine, drools [7]. This is based on the facts supplied from the computing environments, and a dynamically evolving rule base.

This provides an alternative approach to routing messages on the JBI bus, or initiating the delivery of new messages. This can be leveraged to reason on the state of a executing work assignment, using the rule base to make decisions, for example to assist with brokering decisions.

Potentially, a rule engine could be used to orchestrate an entire work assignment. This enables a declarative approach to workflow description. The rules can be changed during runtime opening up some very interesting runtime possibilities such as 'workflow rewriting'. Alternatively, the rule engine could be used at particular points within the course of a workflow execution, such as evaluations at decision points. This hybrid approach using multiple strategies is likely to be the most commonly used.

6 Summary

This paper has reported on some architectural approaches under consideration at the start of the A-WARE project. Clearly, the new breed of grid infrastructure is based on the SOA paradigm. Functional requirements pose a strong need for dynamic message exchanges between all components, which can be added to and removed from the infrastructure in dynamic ways.

A flexible architecture supporting the stated functional requirements is JBI, offering normalized message exchange between components plugged into a message bus. JBI offers general purpose components which will be useful in implementation of A-WARE infrastructure. Very importantly, we envisage re-using many existing opensource libraries for the implementation, writing code to integrate these using JBI. Finally, as a integration framework, JBI offers excellent support for the integration of existing systems and processes.

Work assignments may be described in terms of DSLs, allowing specialists to work in their domain. DSL work descriptions are abstract and will be broken down to concrete submissions of contributing resources. DSLs can be nested and provide a notion of 'canned' workflow. Furthermore, JBI allows for the integration of several orchestration strategies. They can be selected on the basis of particular work assignment. JBI comes with a component supporting BPEL, which can be used as a start, and support for other orchestration engines will be added. A rule engine hosted by JBI declaratively describes consequences of certain states of workflows or events in the environment. Rule engines can potentially be used to orchestrate entire work assignments.

An early prototype of the JBI based framework looks very promising. The great advantage of this approach is the possibility of rapid and flexible development. Development is incremental and highly modular, such that extensions can be added without interfering with the existing components.

References

1. A-WARE Project. http://www.a-ware.org/.
2. AJAX. http://adaptivepath.com/publications/essays/archives/000385.php.
3. Apache ServiceMix. http://incubator.apache.org/servicemix/.
4. Bossa. http://www.bigbross.com/bossa/.
5. Bpmscript. http://www.bpmscript.org/.
6. Dalma. https://dalma.dev.java.net/.
7. Drools. http://drools.codehaus.org/.
8. EnginFrame. http://www.enginframe.com/.
9. European UniGrids Project. http://www.unigrids.org.
10. GGF JSDL. https://forge.gridforum.org/projects/jsdl-wg/.
11. Grid Interoperability Project. http://www.grid-interoperability.org.
12. GridSphere. http://www.gridsphere.org/.
13. Jabber. http://www.jabber.org.
14. Java Business Integration. http://www.jcp.org/en/jsr/detail?id=208.
15. jBPM. http://www.jboss.com/products/jbpm.
16. OSWorkflow. http://www.opensymphony.com/osworkflow/.
17. Portlet Specification. http://www.jcp.org/en/jsr/detail?id=168.
18. D. Erwin, editor. *UNICORE Plus Final Report – Uniform Interface to Computing Resources*. UNICORE Forum e.V., 2003. ISBN 3-00-011592-7.
19. Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. Chair-Richard N. Taylor.
20. Ian Foster. Service-oriented science. *Science*, 308(5723):814–817, May 2005.
21. R. Menday and Ph. Wieder. GRIP: The Evolution of UNICORE towards a Service-Oriented Grid. In *Proc. of the 3rd Cracow Grid Workshop (CGW'03)*, Oct. 27–29 2003.
22. M.Fowler. Language Workbenches: The Killer-App for Domain Specific Languages? 2005. http://www.martinfowler.com/articles/languageWorkbench.html.
23. A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder. Unicore - From Project Results to Production Grids, 2005. Elsevier, L. Grandinetti (Edt.), Grid Computing and New Frontiers of High Performance Processing.