SIMDAT

Data Grids for Process and Product Development using Numerical Simulation and Knowledge Discovery

Project no.: 511438

Grid-based Systems for solving complex problems – IST Call 2

Integrated project

D2.2.2 Report on Grid infrastructure interoperability challenges

Start date of project: 1 September 2004 Duration: 48 months

Due date of deliverable: 01/10/2006
Actual submission date: 01/04/2007

Lead contractor for this deliverable: IT Innovation Centre
Revision: 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination level		
PU	Public	X
PP	Restricted to other programme participant (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Copyright

Table of contents

Executive Summary

This document is the deliverable D2.2.2 “Public report on Grid infrastructure interoperability challenges” of the EU IST-2002-511438 SIMDAT project. The document provides an analysis of key Grid-related interoperability specifications, with a view to defining an adoption policy for industrial Grid developments.

The document outlines the key challenges confronting specifications initiations and vendors wanting to develop compliant infrastructure technologies. Each specification is then described, along with the benefits and considerations associated with their adoption. Following the analysis, Industrial Grid Profile recommendations are defined that will be implemented in successive SIMDAT developments.

WS-Addressing provides a standard way to encode context identifiers, and it is proposed that this be done for all SIMDAT services. However, WS-Addressing also allows a sender to induce a recipient to transmit a (signed) message to a third party, and to include SOAP headers of the sender’s choosing under the signature of the intermediary. This is unacceptable in an industrial Grid environment, and it is proposed that SIMDAT should recognise WS-Addressing specifications for constructing and addressing messages only for a white-list of address and context identifier elements.

WSRF uses WS-Addressing to carry context identifiers for specific server-side resources known as WS-Resources, and specifies an XML document encoding of properties of these resources, plus get, set and query operations on a WS-Resource. These are used to publish WS-ResourceLifetime data, which also supports scheduled or immediate WS-Resource destruction. Finally, WS-ServiceGroup provides a way to create collections of service endpoints characterised by their resource properties, and to search these collections using resource property queries.

Adoption of resource properties requires security considerations: some properties should have restricted access, but the access operations select which properties are returned based on input arguments (query terms), meaning the security policy must be aware of the semantics of resource properties and query languages. Unless security infrastructure exists to is addressed these concerns (e.g. by an argument-aware and query language-aware security infrastructure), it is hard to use WSRF except in a limited way. WS-ResourceLifetime could be adopted but without exposing the lifetime properties (unless they are the only properties of the WS-Resource), but WS-ServiceGroup should not be adopted.

WS-Notification (WSN) defines a collection of interfaces for transmitting notification messages directly between a producer and a consumer using push- or pull-style transfer, plus a specification for distribution of these messages through a broker, and for defining topics that allow subscribers to select particular notifications of interest. These specifications use WS-ResourceProperties, and so depend on how resource properties are secured, although only optional parts of the WSN specifications have this dependency. The main adoption consideration with WSN defining and enforcing access policies for notification subscription and message distribution, to prevent unauthorised access to information through notifications, and possible misuse of notification producers for denial of service attacks.

Finally, OGSA WSRF Profile 1.0 provides a normative profile for implementing Grid services using WSRF and WSN, mandating the use of WS-ResourceProperties and WSN in a wide range of situations. The security policy considerations with WS-ResourceProperties and WSN must be solved for this to make sense in an industrial context. SIMDAT should therefore aim for basic conformance to WSRF initially, and later WSN, using conservative security policies for WS-

ResourceProperties and notification subscription. Each release of the SIMDAT Grid Solution Portfolio will be accompanied by an industrial Grid profile and an explanation of how why that profile was chosen.

1 Introduction

1.1 Purpose

This document is the deliverable D2.2.2 “Public report on Grid infrastructure interoperability challenges” of the EU IST-2002-511438 SIMDAT project.

This document provides an analysis of key Grid-related interoperability specifications, with a view to defining an adoption policy for industrial Grid developments. The purpose is to understand how these specifications can be used by industry and commerce where infrastructure that can support strict but flexible export policies is critical. The document outlines the key challenges confronting specifications initiatives and vendors wanting to develop compliant infrastructure technologies. Each specification is described, along with the benefits and considerations associated with their adoption. Following the analysis, Industrial Grid Profile recommendations are defined that will be implemented in successive SIMDAT developments.

The intended audience for this document are application and technology developers building Grid-based systems, interoperability specifications initiatives, as well as the European Commission Services.

1.2 Scope

Interoperability between Grid infrastructures provided by different vendors is an important requirement for both end-users and application developers. End-users want to be able to dynamically discover and bind to services provided by other organisations without having to be concerned with the underlying Grid technology. Application developers want to be able to provide problem solving environments at a level of abstraction that allows for interoperation between Grid technologies at the service interface without having to maintain complex adaptors. Interoperability is essential if Grid technology is to achieve dynamic and ubiquitous access to heterogeneous resources in a similar way to how the Internet has enabled the WWW.

The Grid and Web Service communities are working towards this ambitious goal through the development of various specifications covering all aspects of service integration such as security, trust, state, notification, etc. In most cases, the specifications are still emerging and changing rapidly with only a few reaching relative stability through the standardisation process. Adoption in industrial production environments is rare and initial deployment, to understand how the specifications can support industrial inter-domain scenarios, is only just beginning.

The deluge of different, complex and sometimes competing specifications has led to various “profiling” initiatives. A “profile” aims to improve interoperability by identifying a group of related specifications that can be used together for a specific purpose and adding further constraints to how the specifications are implemented. This idea originated from a group of leading vendors in the Web Service community called WS-Interoperability (WS-I). For example, WS-I Basic Profile 1.0 specifies that only certain transport protocols should be used (even though WSDL can accommodate others), so that vendors don’t have to implement all possible protocols in their frameworks. Other profiles are now emerging from the Grid community including the OGSA WSRF profile, OGF HPC Profile and NextGRID profiles, however, both are work in progress and compliant systems do not exist today.

In this document we describe the challenges facing businesses aiming to exploit the potential of service-oriented IT infrastructures by looking at the evolution of standards within the Web Service and Grid communities. We then provide an analysis of key Grid-related interoperability specifications including:

- WS-Addressing [1], which describes the encapsulation and use of a (possibly contextualised) Web Service address via End Point References (EPR);
- the WSRF [2] collection of specifications, which describes a particular use of WS-Addressing to access resources via contextualised Web Services;
- the WSN [3] collection of specifications, which builds further on WSRF to define patterns for transmitting notifications between Web Services;
- the OGSA WSRF profile [4], which defines normative functionality expected of an OGSA-compliant Grid, building on WSRF and WSN.

The purpose is to understand how these specifications can be used in industrial, B2B scenarios where infrastructure that can support strict but flexible export policies is critical. Each specification is described, along with the capabilities and adoption considerations. Following the analysis, Industrial Grid Profile recommendations are defined that will be implemented in successive SIMDAT developments.

The analysis provided in this document incorporates conclusions resulting from NextGRID experiments including security issues with WS-Addressing, and scalability considerations with WS-ServiceGroup . In SIMDAT, we have widened the analysis to include a set of specifications that are considered important to industrial Grids, specifically examining the security, operational and dynamic (semantic) requirements of industrial applications.

[1] WS-Addressing, http://www.w3.org/Submission/ws-addressing/

[2] WSRF 1.2 specification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[3] WS-Notification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[4] OGSA WSRF Profile, http://www.ggf.org/documents/GFD.72.pdf

2 Interoperability Challenges

2.1 Background

Today's businesses need to be agile and flexible in order to adapt to changing market conditions and increase their efficiency. The Internet has created new opportunities for business partnerships in a wide range of sectors including engineering, finance, life sciences, media production and healthcare. Grid computing and the underlying service-oriented architectural (SOA) concepts are seen as the solution to making these opportunities a reality. SOA based system design promises to deliver efficient loosely coupled network services, inter-domain business partnerships, and the potential of new business models.

Interoperability between IT infrastructures remains the key challenge to the success of service-oriented businesses. History is beset with failed initiatives aiming to make systems integration between large-scale distributed systems more robust, cost-effective and scalable, for example, Microsoft's DCOM [[5]] and OMG's CORBA specifications [[6]]. Both of these approaches adopted an object-oriented RPC (Remote Procedure Call) communication model and were implemented on various platforms. However, the reality was that unknown clients connecting to services in an inter-domain heterogeneous environment was rarely achieved because applications built on DCOM or CORBA were dependent upon a single vendor's implementation for higher level services (security, transactions, etc). Vendors were competing on implementations capabilities and therefore no motivation existed from a business perspective to achieve interoperability [[7]].

Web Services first appeared in the late 1990s with the focus on making it possible to create, in contrast to CORBA, distributed systems that span organisations connected via the Internet. Web Services are based on a set of open standards supporting the "publish-find-bind" concept from service-oriented architectures. Three core Web Service standards were defined to support this process:

- UDDI [[8]] provides standards for publishing services and for finding them in a directory;
- WSDL [[9]] describes the functionality provided by the service, the message exchanges needed to use it, and (separately) its network address and the transport protocols that must be used for this;
- SOAP [[10]] describes the format of messages, including elements that describe the service function required by the sender, and elements that convey data to and from the service.

In the early days, vendors promised interoperability through the use of vendor, platform, and language independent XML technologies and the ubiquitous HTTP. It should be possible to use any client framework to talk to any service, but in practice, the standards are so rich that this often

[5] DCOM, http://www.microsoft.com/com/default.mspx

[6] CORBA, http://www.omg.org/gettingstarted/corbafaq.htm

[7] "Is Web Services the reincarnation of CORBA?", http://www-128.ibm.com/developerworks/webservices/library/ws-arc3/

[8] UDDI, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3

[9] WSDL, http://www.w3.org/TR/wsdl

[10] SOAP 1.2, http://www.w3.org/TR/soap/

isn't possible. To overcome this, the leading middleware vendors formed a group called WS-Interoperability [[11]], which defines so-called "profiles" – constrained ways to use Web Service standards. For example, WS-I Basic Profile 1.0 [[12]] specifies that only certain transport protocols should be used (even though WSDL can accommodate others), so that vendors don't have to implement all possible protocols in their frameworks. It also specifies that complex types should be represented in WSDL and SOAP messages using XML Schema, so implementers can provide only one mechanism for converting such types to and from messages, etc. The WS-I Basic Profile 1.0 and WS-I Basic Security Profile are now the de facto specifications for developing Web Services.

In addition to these core profiles defined by WS-I, Web Service standards have been proposed to support higher-level services such as trust, federation, state, management, notification, orchestration (workflow), etc. Most are still being discussed at W3C and OASIS. The explosion, conflict and complexity of the specifications, makes interoperability a difficult target for most vendors.

The most significant battle has been to standardise the protocols for resources, events and management. The Globus Alliance with IBM and others launched a new collection of standards called the "Web Services Resource Framework" (WS-RF) in 2004, part of which (concerned with notification) was later decoupled to become "WS-Notification". These proposals were made to OASIS, built on existing and emerging Web Service standards, and were seen as a key step that allows convergence between Web Services and the Grid by supporting applications that require services to support stateful long-running activities. However, although WSRF was ratified by OASIS in Spring 2006 and is compatible with wider Web Services standards (and their likely future development), it was somewhat controversial. Key vendors, Sun and Microsoft, did not back the proposals and went another direction. Sun launched the Web Services Composite Application Framework (WS-CAF) [[13]] which included WS-Context (WS-CTX) for providing a common mechanism for managing and sharing context information between Web Services, whilst Microsoft adopted a similar, but lightweight approach to WSRF, submitting different specifications WS-Transfer and WS-Eventing specifications to the W3C.

The split between the different factions made developing and standardising higher-level application services very challenging. Vendors have to understand the specifications in detail to decide how they support application requirements but importantly they need to make a judgement of the longevity of each initiative. For example, groups working on the OGSA [[14]] are defining a set of specifications profiles for Web Service protocols to support Grid capabilities such as execution management, data transfer, etc. The profiles are largely based on WSRF, however, there are exceptions. For example, OGSA-BES [[15]] does not mandate an underlying resource specification and tries to support both WSRF and WS-I implementations of job services. Building these higher-level application specifications on such a fragile foundation makes interoperability of application level services through standards compliance almost impossible.

Fortunately, the major vendors (HP, Intel, Microsoft, IBM) realised that the dispute needed to be resolved for the greater good and published the WS-Convergence white paper detailing a roadmap

---

[11] WS-I, http://www.ws-i.org/

[12] WS-Basic Profile, http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile

[13] WS-CAF, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

[14] OGSA, http://www.globus.org/ogsa/

[15] OGSA-BES, https://forge.gridforum.org/projects/ogsa-bes-wg/

for converging these specifications over the next 24-36 months (2006 – 2009) [16]. All four vendors promise to deliver products that support these specifications and IBM commits to influence the OASIS WSRF technical committee to refactor the WSRF specifications into extensions that build on WS-ResourceTransfer and the OASIS WS-N technical committee. The white paper does not include Sun and WS-CTX looks likely to fade away.

The road map consists of three sections covering resources, events and management. Figure 1 shows how the existing specifications stack from OASIS and W3C/DTMF will be converged. WS-Transfer and WS-Enumeration is proposed as the base for resource management whilst two new specifications, WS-Transfer Addendum and WS-ResourceTransfer, are proposed. In summary:

- WS-Transfer Addendum will extend WS-Transfer to add support for WS-Addressing endpoint references
- WS-MetadataExchange v1.1 will build on WS-Transfer and WS-Transfer Addendum in place of its current domain-specific protocol
- WS-ResourceTransfer builds on WS-Transfer, WS-Transfer Addendum, WS-Enumeration, and WS-MetadataExchange, providing sophisticated resource management capabilities comparable to those of WSRF

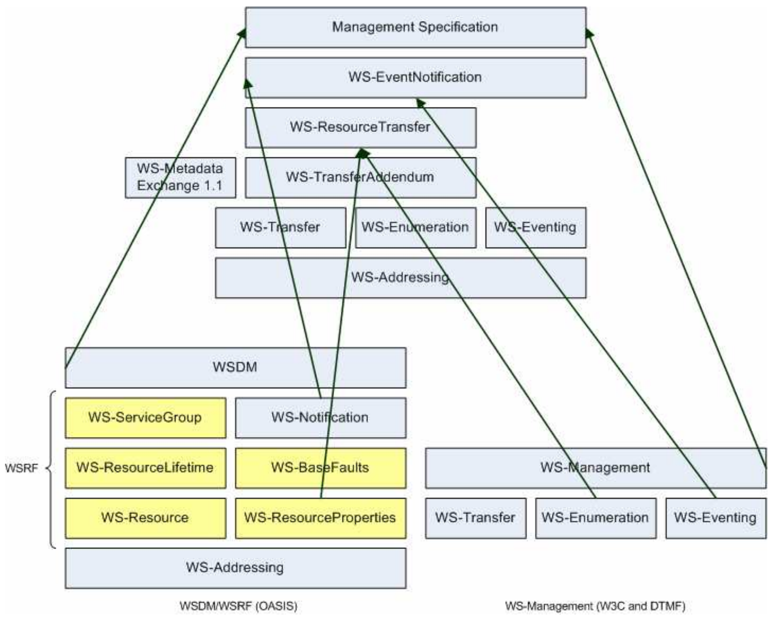

Figure 1: WS-Convergence of resource, events and management specifications

The event processing roadmap proposes using WS-Eventing for event processing. A new specification called WS-EventNotification has been proposed that integrates many of the WS-Notification capabilities. WS-EventNotification builds on WS-ResourceTransfer to support resource management of subscriptions. The management roadmap proposes the development of a

16 "Toward Converging Web Service Standards for Resources, Events, and Management" http://msdn.microsoft.com/library/en-us/dnwebsrv/html/convergence.asp?frame=true

new management specification that builds on WS-EventNotification and WS-ResourceTransfer. The new specification will replace both WSDM and WS-Management [17].

The Web Service and Grid communities are moving slowing towards supporting the ambitious goal of defining open specifications that can support the demands of Internet-based service-oriented businesses. From a technical perspective, Web Service interoperability can be divided into three main layers transport, management, application and content:

- The ***transport layer*** is concerned with protocols for exchanging messages among participants (e.g., HTTP, SOAP, XML).
- The ***management layer*** is concerned with enforcing policies for creating, accessing and monitoring resources in accordance with the terms and conditions of the participants.
- The ***content layer*** is concerned with defining protocols and message structures for accessing information resolving semantic and structural heterogeneity issues of information assets in order to achieve integration of data formats, data models, and languages.

The early initiatives from WS-I focused on the transport layer allowing products from different vendors to exchange XML messages using the SOAP protocol. WS-I is now widely adopted and supported by both commercial and open source products. Interoperability matrices have been published that demonstrate interoperability based on WS-I. Programmers can now develop clients and services in different frameworks that can seamlessly interoperate at the Web Service transport layer; as long as they follow a few simple coding best practices [18].

The focus has now moved to providing open specifications for the management layer building on the work from WSDM/WSRF and WS-Management. The fact that both camps are willing to work together on convergence has similarities to the WS-I initiative and there should be optimism about the group reaching a sensible conclusion even though the timescales are reasonably long.

2.2 The Future Challenges

Reviewing the lessons from history, there are important conclusions we can draw from the Web Service and Grid interoperability efforts over recent years.

The business drivers must be right: It is essential to understand the business drivers behind any interoperability initiative. Businesses are demanding infrastructure that can support Internet-based business partnerships. No single solution will rule and leading technology vendors have recognised the market-potential for products that can work together in underpinning this new economy.

The lessons from CORBA show that vendors were competing on capabilities rather than working towards open specifications. This has similar parallels to the current status of the Grid middleware technologies where many different open source solutions exist each claiming to support inter-domain distributed collaborations. Even though all technologies are now based on Web Services and many on WSRF, the differences in management and application capabilities means interoperability cannot be achieved today. In addition, most solutions are driving for better features (more flexible management, faster data transfer, etc) to define their market position in a period where consolidation may be needed.

[17] “The View from the Bow”, Anne Thomas Manes, March 2006, http://atmanes.blogspot.com/2006/03/ws-convergence.html

[18] http://msdn.microsoft.com/msdntv/episode.aspx?xml=episodes/en/20050210webservicessg/manifest.xml

Many of the challenges for cost effective, secure and robust collaborative working using Grid technologies are still a matter of research even though early adopters within industry are exploring the potential of the technology within their business. The instability of the management layer, a key infrastructure capability for inter-domain collaborations, will cause vendors to base medium term developments on existing commitments to specifications such as WSRF or WS-Transfer. They are unlikely to commit to implementing further interoperability specifications such as WSDM when they will be superseded over the next couple of years by WS-Convergence. The current business value proposition for interoperability is not viable for most Grid vendors who do not want to compromise technological advances to achieve it, considering the level of instability with the current specifications.

Interoperability requires stable standards and specifications: Many different communities are working to solve the challenges of large-scale distributed computing (ebXML, OGF, W3C, etc). Each community is creating proposals that address similar capabilities (trust, security, management, orchestration) which results in standards that evolve and will continue to evolve very rapidly. Hitting such a moving target is a difficult task for vendors trying to use these standards to achieve interoperability between applications and processes. True compliance and interoperability testing can only be achieved when a standard has reached stability. In the face of immature standards a vendor should track the changes, focus on their customer requirements rather than interoperability and implement solutions in the spirit of leading approaches.

Beware of competing proposals from major vendors: When vendors come together to create implementations that embrace the spirit of open specifications and the correct business drivers exist, interoperability initiatives can be successful. Where there are conflicts between major vendors adopting specifications can be very risky and may result in expense re-factoring efforts. We see examples such as the emergence of BPEL from XLang and WSFL and now the WS-Convergence initiative. In such situations, there is usually a compromise allowing each group to demonstrate how earlier initiatives contribute to the new specification(s). In most cases, competing specifications from major vendors can be considered immature rather than de-facto standards. The likelihood of achieving interoperability based on competing specifications is limited. Adoption should be based on using a specification to meet the needs of customer requirements rather than interoperability.

Know the limits of largely academic initiatives: The centre of power in the standards space is with the major software vendors. The influence academic initiatives can have on new standards is limited because of the influence these large vendors have on the overall IT market is very large. Academic initiatives can contribute to the overall progress but these contributions are largely observed by commercial vendors rather than embraced within software products. GGF (now OGF) was a specifications organisation working on supporting the distributed computing needs of the Grid community, largely contributed to by academic organisations. However, few interoperable systems exist today and significant events over the last few years have demonstrated the limits of GGF's power. The initial OGSA/OGSI proposals in 2003 [19] were not accepted by the wider Web Service community and now WSRF will be superseded through commercial vendors actions. The fact that GGF achieved ratification of WSRF by OASIS was an excellent achievement but, although some concepts will remain, the specification is now not considered the solution. The alliance between EGA and GGF to create OGF does bring in more commercial interests into the specifications process but we should wait to see the levels of engagement from commercial vendors and of course only history will judge if specifications developed are successful.

[19] "A developer's overview of OGSI and OGSI-based grid computing" http://www-128.ibm.com/developerworks/grid/library/gr-ogsi/

3 Industrial Grid Profile Analysis

The following section provides an analysis of key Grid interoperability specifications for the purpose of understanding an adoption policy for industrial applications. With the recent WS-Convergence announcement, the management standards space will change significantly over the next few years and existing projects and solutions need to decide on an adoption policy that meets current requirements and does not hold back the evaluation and adoption of Grid technologies by business. Therefore, the following analysis should be considered relevant for adoption over the medium term (12-18 months) whilst the new management specifications emerge.

3.1 WS-Addressing

WS-Addressing was originally designed as a way to convey connection state or context in SOAP messages, emulating the contextualisation mechanism provided by HTTP headers in conjunction with stored cookies. This makes it possible to contextualise message exchanges in a similar way, independently of the transport used.

To support this, WS-Addressing provides a schema for an “endpoint reference” (EPR), which can be used to specify where the response should be sent, and the context headers that should be attached (specified within the endpoint reference through “reference parameters”). The endpoint reference can thus convey a fully qualified and contextualised message destination, and is starting to be used not only to specify the headers needed in a reply, but also to specify onward routing destinations and headers. In this case, the reference parameters are supposed to be treated as opaque by the initial recipient, who is obliged to simply transcribe any reference parameters into the SOAP header of the forwarded message.

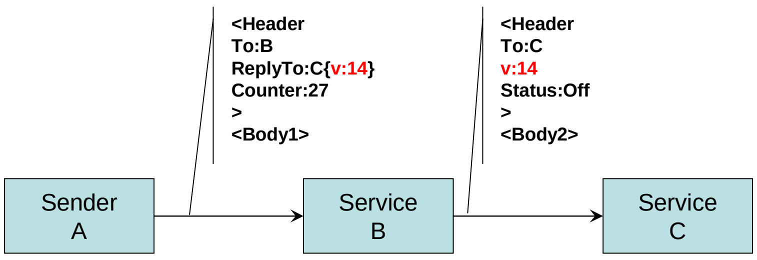

Figure 2: WS-Addressing using ReplyTo with reference parameters

Thus in the above figure the ReplyTo header in the message from A means B should insert the header “<v>14</v>” into its response (abbreviated to “v:14” in the diagram), and send this response to C (not A). Note that B can insert their own headers (e.g. “<Status>Off</Status>”) if desired, but C cannot tell that this header was chosen by B while the other was not.

Several contributors objected to this behaviour, pointing out that this mechanism can force a recipient of a SOAP message to send a message to an arbitrary destination with arbitrary headers. See for example the objection submitted by Anish Karmarkar and others on 19 May 2005, which describes a number of unwanted consequences of opaque header generation in the SOAP binding of WS-Addressing reference parameters. Suppose C in Figure 2 was a service provided by your bank, and B was one of your trusted suppliers. If A sent a message to B including reference parameters “<Account>YourAccount</Account><Debit>£14<Debit/>”, would you want B to construct the WS-Addressing mandate response, sign it, and send it to your bank (Figure 3)?

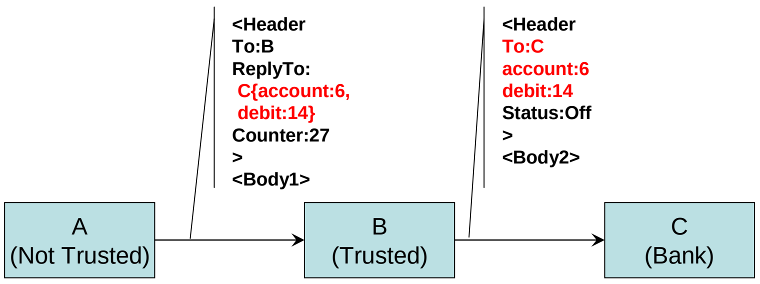

Figure 3: Malicious use of WS-Addressing

Several solutions were proposed, but the one adopted in the Candidate Recommendation of Aug'05 simply mandated that a WS-Addressing type attribute be used to indicate header elements generated from an EPR, and that the "must understand" attribute never be set for such headers. The specification also allows a service to refuse to process an EPR fully if it is not satisfied that to do so would be safe. For example, the specification suggests that this might be determined by authenticating the issuer of the EPR, and processing it only if it comes from a trusted source.

Finally, note that WS-Addressing only forces services to construct a response from an EPR when it is included in a WS-Addressing ReplyTo or FaultTo header element. EPR can also be conveyed to or from a service by other means, in which case the recipient may (but is not required) to use them for contextualised addressing of responses or subsequent messages as though they were sent in the WS-Addressing headers. In practice, this is how WS-Addressing is most often used in Grids. Of course, it doesn't matter how the EPR is transmitted, the problems described here arise whenever an EPR is used to generate context headers for an outgoing signed SOAP message.

Analysis by the NextGRID project has shown that the solution from the August 2005 specification is not satisfactory. Firstly, allowing recipients to ignore WS-Addressing obligations at will means the specification cannot guarantee interoperability. Furthermore, the above attack using reference parameters can also be committed using Address elements in an EPR. For example, the address for C in Figure 2 could be set (by A) to "https://yourbank.com/debit?amount=14", causing B to send a message that achieves the same effect. A may even be able to gain access to protected information by specifying C="http://yourbank.com", causing B to send a message to C without transport-layer security.

Unfortunately, signing over messages and headers does not really solve the problem. If A signs its message to B, this would allow B to tell it came from A, including the ReplyTo header. B may then decide to go ahead and construct a response as specified by A's ReplyTo header, converting reference parameters into full SOAP header elements and sending the resulting message to C. Only B can then bind this header to the message by signing over it, so C can only tell that the response came from B. C cannot tell if B understood the header elements, nor who originally specified them. Thus B's signature can no longer be taken to mean B intended the meaning implied by all parts of its message, but only that B constructed the message to C, including some headers specified by a third party whose identity cannot be verified by C.

If C wanted to know which headers B really did understand, B could insert more attributes, but this is not specified by WS-Addressing so interoperability could no longer be guaranteed. B could insert a second message-level signature covering only the headers it inserted, but this would conflict with the WS-Interoperability Basic Security Profile, which says there should only be one signature to identify the sender of a message. (Multiple signatures would in any case create a semantic interoperability problem, as applications would have to decide the meaning of each signature). B could of course apply a single signature covering only the headers it originated or understands.

However, this would leave the remaining “opaque” headers from A open to tampering en-route between B and C.

Given this, there seems to be only two “reasonable” policies to take regarding the use of WS-Addressing:

- Recipients (B in Figure 2) can constrain where a response generated from an EPR can be sent (e.g. only back to the original sender) so that those who trust them (e.g. C) cannot be induced to act on messages specified by a possibly less trusted third party (e.g. A).

- Recipients can constrain which reference parameters or HTTP arguments from an EPR they are willing to handle, e.g. by using a blacklist of “unsafe” opaque element names, or a whitelist of “understood” element and argument names.

It is also possible to apply both policies at the same time, constraining both the destination and the header content of any response messages dictated by WS-Addressing.

The second policy would allow A to specify some information in the message from B to C, but B would be able to filter out elements B thought unsafe (using a blacklist) or that B did not understand (using a white list). Clearly, a blacklist provides greater interoperability, as services and clients can use any elements not on the blacklist. Using white lists means close coupling between provider and consumer, as consumer needs to know before using a service which reference parameters a service is going to use and service providers cannot easily move to a different set of reference parameters as all clients would need to be notified. However, a blacklist is less secure as it depends on the service operator identifying and blacklisting ALL unsafe elements, or else weakening the guarantees implied by their signature over responses. With a white list, B could sign over the whole response message and signify by this that they understand and vouch for its entire content. It may be significant that in HTTP a white list is used restricting HTTP context headers to a small number of cookie types.

Clearly, if an EPR is sent by some other means (not in a WS-Addressing ReplyTo or FaultTo header), the recipient is not obliged by WS-Addressing to do anything with it. However, the above policies can also be used if such an EPR were used to generate new messages in the conventional fashion.

3.2 Web Service Resource Framework

3.2.1 Overview

The Web Service Resource Framework (WSRF) was the latest attempt by the OGSA to achieve convergence between the Web Service and Grid communities. WSRF is a collection of related specifications intended to define a generic and open framework for modelling and accessing stateful resources using Web Services. WSRF 1.2 has been ratified by the OASIS technical committee. WSRF was developed in partnership with some members of the wider Web Services community and this has helped with acceptance of the proposals outside of the Grid community.

WSRF has been largely adopted by the Grid community with various solutions implementing either part or all of the specifications. However, as discussed in Section 2.1, WSRF was one of several competing proposals for treating state and stateful resources encapsulated in Web Services and, although will remain within Grid solutions for a couple of years, WSRF will be replaced by the results of WS-Convergence.

3.2.2 WS-Resource

The core WSRF specification is Web Service Resource (WS-Resource), which is currently a draft published by OASIS for public review. A WS-Resource is defined as:

- A WS-Resource is the composition of a resource and a Web Service through which the resource can be accessed.

The basic characteristics of a WS-Resource as defined by the specification are as follows:

- references to WS-Resources are represented using WS-Addressing EPR, each of which must refer unambiguously to exactly one WS-Resource;
- the properties of a WS-Resource must be expressed as an XML infoset described by XML Schema and accessible according to the WS-ResourceProperties specification;
- the WS-Resource may support the WS-ResourceLifetime specification; and
- faults generated by a WS-Resource should conform to the WS-BaseFaults specification, and use the WS-Addressing action http://docs.oasis-open.org/wsrf/fault.

The first of these requirements simply defines a profile for using WS-Addressing, and implies that one can encode SOAP messages to address a WS-Resource unambiguously by inserting SOAP headers based on an EPR according to the procedures defined in WS-Addressing. This implies that the WS-Resource can be distinguished by the EPR address element (when only one resource is available at the specified endpoint address), or by the address in conjunction with the EPR reference parameters (which distinguish between different resources available through a single service at the specified address). This is consistent with other uses of message context (including the familiar HTTP cookie mechanism) to indicate that requests should be processed with reference to a specific logical resource managed by a service.

The remaining requirements impose further constraints, which will be discussed in the following sections on the WS-ResourceProperties, WS-ResourceLifetime and WS-BaseFaults specifications.

3.2.2.1 Adoption Considerations

WS-Resource EPRs must comply with the WS-Addressing profile. The need to constrain EPR reference parameters and addresses in WS-Addressing means that the elements used to characterise and refer to WS-Resources must also be constrained.

3.2.3 WS-ResourceProperties

The WS-ResourceProperties specification defines how properties of a resource accessed via a WS-Resource should be described and accessed. The properties must be presented to an accessor in a resource properties document, which is an XML infoset specified as a document type in the WSDL describing the WS-Resource.

The WS-Resource must support access to the resource properties using operations defined in its WSDL, and corresponding WS-Addressing action headers. The operations are shown in Table 1. Each operation is associated with a separate PortType, and apart from GetResourceProperty each operation is optional. The specification goes on to define a way of using WS-BaseNotification and WS-Topics to provide notification of any resource property document changes to subscribing notification listeners.

Operation	**Description**
GetResourceProperty(name)	Returns the resource property specified by the qname "name", or a WS-BaseFault. This operation MUST be supported by every WS-Resource.
GetResourcePropertyDocument	Returns the resource property document, or a WS-BaseFault.
GetMultipleResourceProperties(name+)	Returns the resource properties specified by the list of qnames "name+", or a WS-BaseFault.
QueryResourceProperties(query)	Returns the resource properties whose qnames fulfil the specified "query", or a WS-BaseFault. The query can be described in any suitable language, but all WS-Resource must support XPath queries.
PutResourcePropertyDocument(infoset)	Updates the resource properties document to the supplied "infoset", if possible. Returns an infoset containing the updated elements of the resource property document, or an empty infoset if the update was 100% successful, or a WS-BaseFault.
SetResourceProperties(insert*, update*, delete*)	Updates the resource properties document by inserting, updating or deleting the specified elements. Returns an empty response or a WS-BaseFault.
InsertResourceProperties(insert*)	Updates the resource properties document by inserting the specified elements. Returns an empty response or a WS-BaseFault.
UpdateResourceProperties(update*)	Updates the resource properties document by updating the specified elements. Returns an empty response or a WS-BaseFault.
DeleteResourceProperties(delete*)	Updates the resource properties document by deleting the specified elements. Returns an empty response or a WS-BaseFault.

Table 1: WS-ResourceProperties interface

3.2.3.1 Adoption Considerations

Resource properties break the normal encapsulation patterns found to be most successful in e-Commerce Web and Web Service applications. Resource properties provide semantically opaque access to stored attributes of a resource**.** Allowing consumers to access WS-ResourceProperties exposes them to possible interoperability problems when their service providers make changes, which can lead to application fragility. One solution to the application fragility problem is to simply deny access to all resource properties except where:

- the resource property in question is defined by a stable and widely adopted Web Service specification, such that all clients and providers can be expected to understand it; or

- the request comes from an application that is maintained together with the WS-Resource, so that changes in resource properties can be handled by changing the application.

Fundamentally, service providers should not change syntax and semantics of resource properties that have been published to clients unless these changes can be encapsulated. Note that the second criterion is not met if the application and the WS-Resource are written by the same organisation, since different deploying organisations are unlikely to upgrade at the same instant. The application must normally also be deployed by the same organisation as the WS-Resource. In effect, this means WS-ResourceProperties should only be accessed by the service provider hosting the WS-Resource.

The specification does not exclude concurrent access to a resource properties document. The specification points out that it may be necessary to define transactional characteristics for concurrent access, but does not say how this should be done. This means that a requesting application must be aware of the semantics and representation of the properties of a resource, including any mechanisms for resolving concurrent access conflicts, in order to use and set resource properties correctly. This is important for all Web Services that have multiple operations updating stateful resources, however, existing Web Service specifications work on the principle of "stateless" interactions whereas WSRF is explicitly stateful and should address transaction concerns.

Access control restrictions for resource properties are undefined. If any of the stored attributes of a WS-Resource are subject to access control restrictions, then the security mechanisms to enforce this must also understand the semantics and representation of the properties, and possibly also of any query language used to retrieve them. Since all resource properties are accessed via the same WSDL operations the security infrastructure must take into account all elements of the request to work out which properties were being requested before deciding whether the request is authorised. No implementations exist to achieve this in a service-independent way. The only simple way to support different access constraints for each resource property is to support only the mandatory GetResourceProperty operation, so applications cannot insert or change the values of resource properties, and so that the target of each request is unique and easily extracted from the request document. Other WS-ResourceProperties access methods cause complications, and the most general QueryResourceProperties method, which can process arbitrary queries, is the most difficult to deal with. As no general infrastructure policy implementations exist for this operation it can only be secured by applying the policy in the method implementation either through a database view or by filtering the result set after the query has been executed

To address the encapsulation and access control problems identified above in a simple way, the use of WS-ResourceProperties must be constrained. There are three basic options:

- specify that the "conversational" resources can have no resource properties, so that all the methods return faults in all cases;

- allow access to the mandatory GetResourceProperty method of WS-ResourceProperties only, with a fixed set of "well-known" properties for each WS-Resource, and an access policy for each property; or

- restrict access to WS-ResourceProperties to the host provider of the WS-Resource, thus containing any interoperability problems, and allowing a simple, uniform security policy for all resource properties and access methods.

The first of these options is the simplest, but also the most restrictive as it precludes the use of other specifications that build on the wider WS-ResourceProperties. The second option works by preventing resource property insertions (so that a fixed access policy for well-known properties can be used), but this also prevents the use of specifications that depend on manipulating the resource property document. The last option allows all of WS-ResourceProperties to be used, including specifications that build on this, but only within the host provider of the WS-Resource.

3.2.4 WS-ResourceLifetime

The WS-ResourceLifetime specification defines a mechanism for destroying a WS-Resource, either on request by a client or at a scheduled termination time:

Operation	**Description**
Destroy	Returns an acknowledgement, or a WS-BaseFault if the resource is not destroyed or is unknown to the service.
SetTerminationTime(time | duration)	Returns an acknowledgement, or a WS-BaseFault if the termination time of the resource cannot be changed, or the resource is unknown to the service.

Table 2: WS-ResourceLifetime interface

Each operation makes up its own PortType, and both operations are optional. In addition, a WS-Resource that supports WS-ResourceLifetime must provide two read-only resource properties:

- the current time kept by the WS-Resource; and
- the current termination time for the WS-Resource (with a Boolean attribute indicating if the WS-Resource has no set termination time).

3.2.6.1 Adoption Considerations

Accurate clock comparison mechanisms are not defined. The “current time” resource property is provided so that a client can interpret the “current termination time” even if the clocks of the client and WS-Resource service are not synchronised. However, this can only be used to detect very gross discrepancies, due to the inevitable (and variable) network delays. WS-ResourceLifetime does not specify any mechanism for an accurate clock comparison.

WS-ResourceLifetime depends on the adoption policy for WS-ResourceProperties. The usefulness of WS-ResourceLifetime is constrained according to the option chosen for addressing problems with resource property access (See section 3.2.3):

WSRF-RP Option	**Impact on WS-ResourceLifetime**
No resource properties.	WS-ResourceLifetime could not be supported, as the mandatory current time and current termination time properties would not be available. It would still be possible to adopt the WSDL port types for operations to destroy a WS-Resource, but extra operations would be needed if users require access to the current lifetime properties of a WS-Resource.
Properties accessible to the host provider only.	WS-ResourceLifetime could be adopted in full, but the current lifetime properties would not be accessible outside the host provider. Extra operations would be needed if users require access to the current lifetime

	properties of a WS-Resource.
Read-only access rights defined per property.	WS-ResourceLifetime could be adopted in full.

Table 3: WS-ResourceLifetime adoption options

WS-ResourceLifetime does not support the full resource lifecycle. The WS-ResourceLifetime is also said to support resource lifecycles, but this is not so. It only provides a set of destruction mechanisms, and a way of obtaining the (approximate) scheduled termination time. Some "custom" WSDL methods will usually be needed to support other lifetime events such as WS-Resource creation.

3.2.5 WS-ServiceGroup

The WS-ServiceGroup specification defines a Web Service that maintains heterogeneous information about a collection of WS-Resources. The main use of WS-ServiceGroup is the implementation of a soft-state registry of Web Services, in which WS-Resources can also register their resource properties. A summary of service group concepts are given below:

- ServiceGroup (identified by a ServiceGroupEPR) is a WS-Resource that aggregates information through resource properties operations from a collection of member Web Services, that themselves can be WS-Resources;
- ServiceGroupEntry is a WS-Resource describing the association of a member Web Service (identified by a MemberEPR) with a Service Group;
- ServiceGroupEPR is an EPR referring to the ServiceGroup to which the member Web Service belongs;
- MemberEPR is an EPR referring to the member Web Service itself;
- Content contains elements describing the membership of the Web Service in the ServiceGroup.

If the member Web Service is itself a WS-Resource, the Content resource property may include an RPDoc child element quoting the resource property document of the member service. Note that in this case, the contents of the RPDoc child element should track any changes in the member service's resource property document.

The WS-Resource providing access to a ServiceGroupEntry must provide access to its properties. It should also support WS-ResourceLifetime (allowing the destruction of ServiceGroup membership associations) and the NotificationProducer interface from the WS-BaseNotification specification with a WS-Topic for notifying changes in the Content resource property.

The ServiceGroup provides resource properties describing the group and its members:

- MembershipContentRule: specifies as attributes the list of interfaces that must be supported by members of the ServiceGroup, and a list of XML Schema global element declarations that must be present in the Content resource property of the ServiceGroupEntry for each member.

- Entry: one per member service, providing the contents of the ServiceGroupEntry resource properties document for that member.

The WS-Resource providing access to the ServiceGroup should also support WS-ResourceLifetime, and implement a policy for the destruction of associated ServiceGroupEntry resources when the ServiceGroup itself is destroyed.

Operation	**Description**
Add(member, content, [timeout])	Creates a ServiceGroupEntry for the service referred to by the member EPR, with an associated Content given by the content argument, and an optional termination time specified by timeout (if present). If successful, the ServiceGroup should respond with an EPR for the ServiceGroupEntry created, along with its termination time and the ServiceGroup's current time.

Table 4: Service Group registration interface

A ServiceGroup WS-resource may be extended by adding the ServiceGroupRegistration operation (see Table 4). If successful, the ServiceGroup should respond with an EPR for the ServiceGroupEntry created, along with its termination time and the ServiceGroup's current time. There is no "Remove" operation, presumably because this can be implemented by destruction via WS-ResourceLifetime methods of the corresponding ServiceGroupEntry resource, or of the ServiceGroup itself. Finally, a ServiceGroup may optionally support WS-BaseNotification to provide notification of changes in its composition. In that case, it is obliged to support EntryAdditionNotification and EntryRemovalNotification topics using message types defined by WS-ServiceGroup.

3.2.5.1 Adoption Considerations

WS-ResourceLifetime depends on the adoption policy for WS-ResourceProperties. WS-ServiceGroup does not specify a query mechanism but uses the QueryResourceProperties mechanism from the WS-ResourceProperties specification that is used to submit arbitrary queries on the resource properties document of the ServiceGroup. Consequently, WS-ServiceGroup functionality depends on the adoption policy for WS-ResourceProperties:

WSRF-RP Option	**Impact on WS-ServiceGroup**
No resource properties.	WS-ServiceGroup would be unusable.
Properties accessible to the host provider only.	WS-ServiceGroup could be supported in full, but only co-located WS-Resources could be registered, and queries could only be submitted by the host provider.
Read-only access rights defined per property.	WS-ServiceGroup can only be implemented if a security handler existed that could enforce policies for complex queries targeted at the ServiceGroup resource properties documents

Table 5: WS-ServiceGroup adoption options

Enforcing a security policy for service group content based on member's policies cannot be easily achieved. The service group policy infrastructure would need to enforce access restrictions on service group content. If a policy infrastructure could be devised to understand the semantics of QueryResourceProperties, requests policy decisions could be enforced at the service group. However, it would be extremely difficult to enforce the security policies of member WS-Resources, unless the service group policy was also dynamically derived from member policies.

Synchronisation strategies between service group content and member's resource properties documents are not specified. WS-ServiceGroup does not specify how content aggregated from a collection of Web Services should be synchronised, or how far out of date a ServiceGroupEntry may become, or what mechanisms (if any) should be used to minimise problems caused by any delay.

3.2.6 WS-BaseFault

The WS-BaseFault specification defines an XML Schema type for base faults, along with rules for how this base fault type is used and extended by Web Services. The BaseFault type contains the following:

- a single, mandatory Timestamp element giving the time at which the fault occurred;
- an optional OriginatorReference element giving the WS-Addressing EPR for the generating service;
- an optional ErrorCode element providing a legacy error code, and a dialect attribute URI denoting how this legacy code should be interpreted;
- one or more Description elements each carrying a plain language string intended to be intelligible to the user;
- an optional FaultCause element containing another BaseFault describing the underlying cause, allowing a causal chain of faults to be described;
- extension elements as required by the implementor.

A BaseFault does not include any "type" attributes, so faults that have distinct semantics must be described by extending the BaseFault type. It is intended that an operation may only return faults that have the BaseFault type defined in the WSDL, or a more refined type.

The example SOAP 1.2 encodings indicate that a BaseFault message should be encoded in the Detail element of a SOAP fault. The BaseFault therefore provides a way to classify SOAP faults and endow them with some meaning that can be related to the faulting service.

3.2.6.1 Adoption Considerations

Existing legacy services need to be re-factored or wrapped. The only problem with WS-BaseFault is that all the WSRF specifications including WS-Resource indicate that a WS-Resource must respond with a BaseFault if an error occurs. This means that one cannot easily add WSRF functionality to a legacy service that uses other types of SOAP faults.

3.3 WS-Notification

3.3.1 Overview

The WSN family of specifications were originally developed to fulfil the need identified by many members of the Grid community for notification messages to be passed between components of a Grid infrastructure. This requirement was originally motivated by the fact that Grid applications tend to be computationally complex, and tend to require long execution times even on high-end computing platforms. Short-lived computations can easily be wrapped in a single remote procedure call, using the TCP/IP connection to carry the call and response between the invoker and the service. However if a long-lived Grid applications is wrapped in this way, the interval between the call and response is likely to exceed the lifetime of the TCP/IP connection. This may simply be because the calling application cannot be left running that long, but it is also common for the intervening network to kill connections if there is no activity for a reasonable period, to prevent "dead" connections using up all the assignable ports.

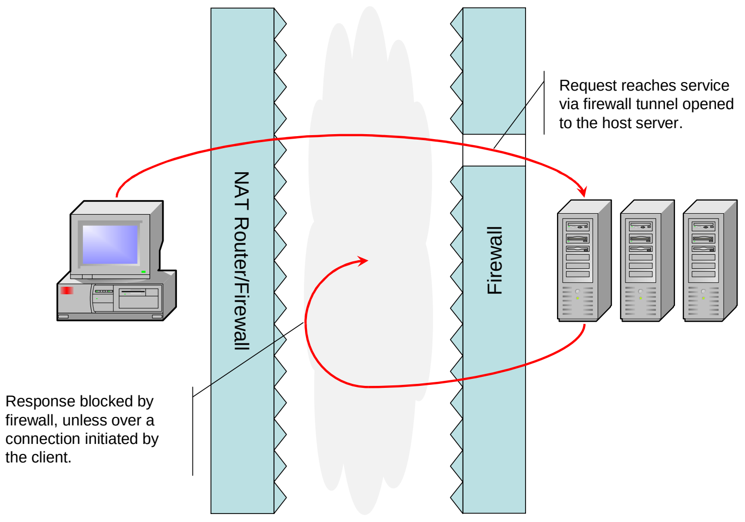
Request reaches service via firewall tunnel opened to the host server.
NAT Router/Firewall
Firewall
Response blocked by firewall, unless over a connection initiated by the client.

Figure 4: Firewall interference in UDP notification traffic

The original solution to this problem, which dates back to early distributed computing over local area networks, was to pass notification messages from the service to the invoker using a datagram transport such as UDP. This has the advantage that a stateful connection is not required between the two ends, so the traffic is less likely to fail because of client application or network timeouts. The down side of this approach is that the client and service must be mutually addressable over IP, and willing to accept UDP packets sent direct from one to the other. In practice, services can usually be configured to allow this, but client application users are often behind firewalls, as shown in Figure 4.

Typical firewalls are configured to allow internal machines to initiate connections to external servers, often using Network Address Translation so the client doesn't even need an Internet-accessible address. Because such machines cannot even be addressed from outside their local network, it isn't possible for the server to send back a response unless it uses an existing connection set up by the client. Most firewalls also block incoming connections even to machines that have Internet-reachable addresses, except to well-maintained servers that provide standard services like email and web applications.

This is one of the reasons why Grid users often have difficulty making the software work. The typical end user is not in a position to have an Internet-accessible address assigned to their machine and firewalls and other gateway devices configured to allow external access to it. If this is done, their machine will certainly be subject to countless, mostly automated intrusion attempts. Typical network administrators are not in a position to ensure these user-managed systems remain secure against such attacks, which means they become weak points in the organisational security infrastructure, and access normally has to be closed down if an intrusion is suspected.

The migration of Grid infrastructure to a Web Service platform was partly motivated by the idea that web traffic would be more “firewall friendly”. However, opening any direct access to a user’s workstation carries considerable risk, and cannot be done even for web traffic if the user’s machine accesses the Internet via a NAT device. The WSN specifications are being developed within OASIS to standardise the pattern of message exchanges needed to send notifications between systems using Web Services, and to provide ways to do this that can work in both directions even between NAT and firewall-protected systems.

3.3.2 WS-BaseNotification

3.3.2.1 Scope

The simplest WSN specification is Web Services Base Notification (WS-BaseNotification) which was approved by the OASIS technical committee in July 2006. The specification defines a “publish-subscribe” pattern for transmitting notification messages between a NotificationProvider and a registered NotificationConsumer.

3.3.2.2 Push-style notification

The NotificationConsumer interface defines a Notify message structure, which can contain one or more NotificationMessage elements, each containing the following information:

- ➢ an optional subscription reference giving the EPR of the service through which the NotificationConsumer registered an interest in this notification;
- ➢ an optional topic element, defining the topic (and dialect) to which this notification message is related;
- ➢ an optional producer reference, giving the EPR of the WS-Resource that generated the notification;
- ➢ a message element containing the notification itself, whose content is application related and not defined in the specification;
- ➢ further elements that may be inserted to support extensions to WS-BaseNotification.

A notification consumer may choose to support “raw” notification, in which application specific messages are delivered to it in their native format, or a WSDL operation to handle receipt of the Notify message format (with no response), or both.

Notification producers produce notification messages, and must support the NotificationProducer interface, which provides two operations as shown in Table 6. In addition, the NotificationProducer may support the WS-ResourceProperties specification, and if so make available the following ResourceProperties:

- ➢ zero or more TopicExpression elements, indicating topics supported by the producer (these are recommended to conform to the WS-Topics specification);
- ➢ zero or more TopicExpressionDialect elements, indicating the topic dialects supported;

- an optional FixedTopicSet element, indicating whether the set of supported topics may change (assumed not to be the case if this element is missing).

The idea here is that when a NotificationConsumer subscribes to a NotificationProducer, the producer will set up a Subscription WS-Resource and send notifications to the specified end point, for a specified time, subject to the specified policy and filter. These notifications are delivered to the NotificationConsumer in Notify messages, unless the policy includes the UseRaw element, in which case the notifications are sent "as is", with no wrappers to the consumer defined by the supplied endpoint reference.

It is also possible for other clients to access the current or last notification message for a given topic, but this does not constitute delivery of the message, and does not interrupt its transmission to registered consumers.

Operation	**Description**
Subscribe(consumer, [filter], [policy], [time])	Registers interest from a NotificationConsumer at the EPR "consumer" to receive notifications that match the pattern specified in "filter" until the specified (absolute or relative) "time". The filter can refer to notification topics, other message elements, or the ResourceProperties of the producer. The "policy" argument defines other constraints, such as the notification message rate, and whether notification messages should be wrapped in the Notify schema, or delivered "Raw". The response contains the EPR for the Subscription resource that will deal with the registered consumer. If the EPR supports destruction via WS-ResourceLifetime, then the response should also contain the current time and the termination time.
GetCurrentMessage(topic)	Retrieves the last notification published on the specified topic. This is a non-destructive read, designed for use by actors other than subscribed consumers. The NotificationProducer may choose not to retain or provide the last notification, in which case it should return a fault message indicating no current message is available on that topic.

Table 6: WS-BaseNotification interface

3.3.2.3 Pull-style notification

WS-BaseNotification goes on to specifies a pull-style message delivery mechanism, designed for use by consumers to which notifications cannot be delivered, e.g. because they are behind a firewall. Two interfaces are defined to support this: the CreatePullPoint port type and the PullPoint port type. The CreatePullPoint interface supports a method for creating a PullPoint as shown in Table 7.

Operation	Description
CreatePullPoint()	Creates a pull point, returning an EPR for the pull point itself.

Table 7: CreatePullPoint interface

The specification says that the creation request must be sent to an end point that supports the PullPoint interface, suggesting that both interfaces are supposed to be combined in one service. This would imply that the CreatePullPoint would return the EPR of that service, though possibly with a different reference parameter to contextualise the PullPoint interface. The specification thus appears to rule out the possibility that somebody (e.g. a firewall administrator) may wish to set up several PullPoint facilities perhaps using different transports (HTTP, SMTP, IMAP, etc), and provide a separate CreatePullPoint service through which users can be assigned to different types of PullPoint depending on their role, status or security clearance.

In any case, the PullPoint obtained should support the NotificationConsumer functionality described above, through which it receives notification messages that need to be delivered. The PullPoint interface then specifies two further operations as shown in Table 8.

Operation	Description
GetMessages(N)	Retrieves up to N notification messages from the pull point interface, if available. The response contains a collection of NotificationMessage elements, similar to the Notify message used for push-style delivery. Messages retrieved from a pull point are considered delivered to the client consumer.
Destroy()	Terminates the PullPoint resource.

Table 8: PullPoint operations

The workflow for using a PullPoint is not spelled out by the specification, but it seems clear that a user should first obtain a PullPoint EPR, and pass this as the “consumer” field in a Subscribe message to a NotificationProducer. This will deliver any notifications to the PullPoint, and the user can retrieve them from there using the GetMessages operation.

The specification also says a PullPoint may or may not be a WS-Resource, but if it is it must support the WS-ResourceLifetime immediate destruction, and may support scheduled destruction.

3.3.2.4 Subscription management

At this point, it seems that the functionality of WS-BaseNotification is complete: if a NotificationProducer exists, consumers can subscribe to it and retrieve a Subscription EPR, using a PullPoint if required. The PullPoint may not be a WS-Resource, but it can be destroyed using the WS-BaseNotification Destroy operation in any case. The Subscription is a WS-Resource, so that can be destroyed when the subscription is no longer needed. In addition to all this, any client can retrieve the current (i.e. last) message from a NotificationConsumer. However, WS-BaseNotification also specifies subscription management services, apparently so that (like

PullPoints) Subscriptions need not be WS-Resources, even though the NotificationConsumer specification suggests that they should be.

The SubscriptionManager interface is divided into two parts; the BaseSubscriptionManager (See Table 9) and the optional PausableSubscriptionManager (see Table 10) interfaces:

Operation	**Description**
Renew(time)	Allows a requestor to specify a relative or absolute termination time for a subscription. The response includes the subscription manager's current time and the new termination time of the subscription.
Unsubscribe()	Allows a requestor to instruct the subscription manager to terminate the subscription resource. If it can't do this it must throw a fault.

Table 9: BaseSubscriptionManager interface

Operation	**Description**
PauseSubscription()	Allows a requestor to pause a subscription. This means notifications will not be delivered to the consumer as long as the subscription is paused.
ResumeSubscription()	Allows a requestor to resume a subscription that was previously paused. This means notifications will start being delivered again. The producer may then deliver all notifications or just the last notification that arose while the subscription was paused, or simply resume delivery with the next notification that arises.

Table 10: PausableSubscriptionManager interface

The Renew and Unsubscribe operations reproduce the functionality of WS-ResourceLifetime, but presumably this is not used because (a) Subscriptions are not obliged to be WS-Resources, and (b) the BaseSubscriptionManager operations are not optional.

If Subscriptions are WS-Resources, then they are obliged to include the following resource properties:

- ConsumerReference: the EPR of the NotificationConsumer specified in the subscription;
- Filter: the filter on which notifications are to be sent to the consumer;
- SubscriptionPolicy: the policy defining the rate of notifications, etc;
- CreationTime: the time at which the subscription was created.

The values of these resource properties are initially set by the NotificationProducer that created the Subscription (in response to a Subscribe request). They may subsequently be changed via WS-ResourceProperties update operations, if the Subscription supports this.

3.3.2.5 *Adoption Considerations*

Avoiding dependency on WSRF produces ambiguities in the specification. The specification states that a PullPoint may or may not be a WS-Resource, but if it is it must support the WS-ResourceLifetime immediate destruction, and may support scheduled destruction. This duplicates the functionality of the WS-BaseNotification Destroy operation, which is not the same as the Destroy operation in WS-ResourceLifetime (the messages are distinct). This may be because support for destruction is mandatory, while Destroy is an optional method in WS-ResourceLifetime.

For the SubscriptionManagement interface it is clear that the operations relate to a particular Subscription. Presumably, a SubscriptionManager service relates to a single Subscription service, or to a collection of Subscriptions accessed via contextualised requests, as would be the case if Subscriptions were WS-Resources. However, the specification does not explain how contextualisation could be achieved if Subscriptions are not WS-Resources.

It is also possible that the developers of WS-BaseNotification wanted to avoid making the use of WSRF mandatory for implementers, although this has not been done in all parts of the specification.

If implementations depend on WSRF then WS-Notification depends on the adoption policy for WS-ResourceProperties. Although, WS-Notification states that producers and consumers do not have to be WS-Resources, if they are then WS-ResourceProperties need to be considered.

WSRF-RP Option	**Impact on WS-BaseNotification**
No resource properties.	WS-BaseNotification could be used, but with restrictions on some of the optional functionality: NotificationProducers could not expose the topics on which they can produce notifications as resource properties. Subscriptions could not use resource properties to expose subscription details or support WS-ResourceLifetime. PullPoints could not be WS-Resources, or use WS-ResourceLifetime.
Properties accessible to the host provider only.	WS-BaseNotification could be used "in full" between co-located services hosted by the same service provider, but would otherwise be subject to the above restrictions.
Read-only access rights defined per property.	WS-BaseNotification could be used, except for the optional functionality for changing subscription details by updating exposed Subscription resource properties.

Table 11: WS-Notification resource properties adoption

Security policy enforcement strategies to protect against inappropriate information disclosure or denial of service attacks via notification messages need to be considered. WS-BaseNotification includes the usual remarks about the need for security policies to protect WS-Resources. This means access to the Subscribe operations of a NotificationProducer should be protected by a security policy that prevents an unauthorised person gaining access to information by means of notification messages. It is important to realise that the Subscriber who calls the NotificationProducer Subscribe operation need not be the same entity specified as the corresponding NotificationConsumer. This provides an indirect mechanism whereby information could be leaked to an unauthorised party via notification. It also makes the NotificationProducer potentially exploitable as a denial of service agent, by subscribing to receive very frequent notifications, and providing the DoS target's EPR instead of a legitimate consumer. Since the notification messages do not have a response, and need not even be wrapped in a Notify message giving the originator's EPR, the target would be unable to tell the NotificationProducer that its notification messages are not welcome.

Given this, the current WS-BaseNotification specification is dangerous. It is clearly necessary to restrict access to the Subscribe function of a NotificationProducer, but defining a policy for this is non-trivial, since:

- all subscription attempts go via the same WSDL operation, so it is not possible to discriminate via simply defined and enforced constraints on access to this operation;
- the information revealed in notifications depends on the (potentially complex and optional) filter argument – the interface doesn't use a simple identification of subscription topics;
- the destination may not be acceptable to the NotificationProducer, even if the Subscriber were accepted – e.g. they may have been induced to specify a consumer EPR provided by a malicious third party.

It is also difficult to enforce a suitable policy, even if one could be defined. Any enforcer would have to inspect the whole subscription request, and decode the implications of a request (including the filter) in order to determine whether access should be granted. This means the policy could not reasonably be enforced by a service-independent security layer, but would have to be (at least partially) encoded by the specific NotificationProducer implementer.

The only simple constraint that could easily be implemented is to insist that subscriptions can only be established, and notifications only be delivered to services hosted by the same provider as the NotificationProducer. If this constraint is imposed, it would also be possible to adopt a similar constraint on the use of WS-ResourceProperties, allowing all WS-BaseNotification functionality to be used within the service provider environment.

3.3.3 WS-BrokeredNotification

The Web Services Brokered Notification 1.3 specification builds on WS-BaseNotification to define an intermediate notification broker that acts as both a NotificationProducer and a NotificationConsumer. The idea is that the NotificationBroker subscribes (as a consumer) to notifications from other producers, and accepts (as a producer) subscriptions from other consumers to which it distributes the notifications it receives. This provides two main benefits:

- it improves scalability: if N consumers want to receive notifications from P producers, the NP direct interactions between them can be replaced by N + P interactions with the broker;
- it allows notification topics to be organised into hierarchies by the broker, making it easier to subscribe to complex subsets of the notification messages produced.

WS-BrokeredNotification is intended to support enterprise-scale notification messaging using a "publish-subscribe" pattern similar to that found in other technologies such as JMS [[20]].

[20] JMS

The NotificationBroker specification asserts that notifications will originate from Publishers, which may be NotificationProducers supporting the Subscribe functionality, but may also be arbitrary sources of raw notifications. Notifications will be sent to NotificationConsumers who have subscribed to the relevant topic(s). The NotificationBroker acts as a go-between, and must support the following interfaces:

- the Notify message exchange patterns for receiving notifications from Publishers and for sending notifications to NotificationConsumers;
- the Subscribe and GetCurrentMessage operations of a NotificationProducer, allowing end-consumers to subscribe to receive messages via a consumer EPR;
- the CreatePullPoint operation, allowing end-consumers to create a PullPoint for "passive" message retrieval.

The last of these implies a stand-alone CreatePullPoint capability, even though the WS-BaseNotification specification implies this must always be combined with the PullPoint interface. In fact, the specification is clear that a NotificationBroker need not support PullPoint message retrieval, but it must support CreatePullPoint and respond with a fault if it does not also support the PullPoint interface.

In addition, a NotificationBroker must support the RegisterPublisher interface as shown in Table 12.

Operation	**Description**
RegisterPublisher([publisher], [topics], [demand], [time], …)	Registers a publisher (EPR), providing notifications on a list of topics until a specified initial termination time. The response is a PublisherRegistrationManager (EPR) that includes a reference parameter corresponding to the specific PublisherRegistration created.

Table 12: RegisterPublisher interface

There are two models for the interaction between a Publisher and NotificationBroker once the publisher has registered. In the simple model, the publisher simply starts to transmit notification messages to the broker, allowing it to forward them to any of its consumers that subscribe to it to receive these notifications. The alternative is the demand-based model, specified through the optional "demand" registration argument, in which the publisher does not start sending until the broker has subscribed to it as a consumer. Whenever the broker has no consumers for the specified notifications, it sends a PauseSubscription message to the publisher, so that the publisher doesn't have to send notifications to the broker when the broker has no customers that want them. Note that if the Publisher does not specify a "publisher" EPR when it registered, it means it is not a NotificationProducer or does not wish to receive messages from the NotificationBroker. In either case it may not specify the demand-based model on registration.

The PublisherRegistrationManager then supports a further operation:

Operation	**Description**
Destroy	Destroys a PublisherRegistrationManager.

Table 13: PublisherRegistrationManager interface

It appears that a PublisherRegistration may be a WS-Resource, and may support destruction via WS-ResourceLifetime. However, as with the Subscription resource in WS-BaseNotification, the destruction operation is mandatory, and so is provided by the PublisherRegistrationManager. The Destroy operation is supposed to destroy the PublisherRegistrationManager, but the intention seems to be that it also terminates the associated PublisherRegistration.

The PublisherRegistration may be a WS-Resource, and so may support WS-ResourceProperties access requests for the following Resource Properties:

- an optional PublisherReference, giving the EPR of the registered publisher;
- zero or more Topic elements, giving the topics for notifications produced by the publisher;
- a Demand element indicating whether the registered publisher uses the demand-based model of interaction with the broker; and
- an optional CreationTime, giving the time the publisher registered.

3.3.3.1 Adoption Considerations

Avoiding dependency on WSRF produces ambiguities in the specification. The specification isn't very clear about the relationships between a PublisherRegistrationManager and a PublisherRegistration. This may be because the authors see the PublisherRegistration as a WS-Resource accessed via a PublisherRegistrationManager service, but want to avoid making WSRF mandatory for implementers.

If implementations depend on WSRF then WS-Notification depends on the adoption policy for WS-ResourceProperties. WS-BrokeredNotification is constructed from WS-BaseNotification specifications, except for the RegisterPublisher interface, and the PublisherRegistration and PublisherRegistrationManager services and functionality. Therefore, like WS-BaseNotification, WS-BrokeredNotification does not require the use of WSRF, but there is some loss of optional functionality if WS-ResourceProperties are constrained.

WSRF-RP Option	**Impact on WS-BrokeredNotification**
No resource properties.	PublisherRegistrationManagers cannot expose resource properties giving the registered publisher, creation time, interaction model and topics, or use WS-ResourceLifetime. The NotificationBroker will also be unable to use some optional functionality of WS-BaseNotification, e.g. exposing NotificationProducer topics as resource properties, etc.
Properties accessible to the host provider only.	WS-BrokeredNotification could be used "in full", but only for sending notifications between co-located services hosted by the same service provider.
Read-only access rights defined per property.	PublisherRegistrationManagers could not use resource properties to allow update of their publisher registration details.

Table 14: WS-BrokeredNotification resource properties adoption

Security policy enforcement strategies to protect against inappropriate information disclosure or denial of service attacks via notification messages need to be considered. As with WS-BaseNotification, the WS-BrokeredNotification specification includes a discussion of security. The specification demands that a broker should only allow registration by authorised publishers, should only accept messages of the registered types from these publishers, and should prevent any modification or termination of PublisherRegistration resources except by the authorised principals. In addition, it may control which producers can publish on a topic, impose security measures on messaging routed through the broker, and provide security management based on topics. However, it does not say how any of these should be implemented.

The specification does discuss the need to manage access to notifications, and suggests that this can be done by using a topic hierarchy. However, as we have seen, it is very difficult to enforce subscription policies that depend on topics in a general way. A more sensible option is for each NotificationBroker to decide which topics they wish to handle, given the access criteria they impose on subscribers. The NotificationBroker should then refuse publisher registrations and drop notification messages for topics they don't wish to handle.

The main problem with this is that publishers have to trust the NotificationBroker to refuse registrations or messages that their subscribers should not see. For this reason it may be appropriate for publishers to use only co-located NotificationBrokers. This constraint must be implemented by the publishers, as if a publisher doesn't trust the consistency of a broker's topic handling and subscriber policies, then it shouldn't trust the broker to refuse registrations from remote publishers.

Note that there is no risk that a publisher could be forced into a DoS attack if the NotificationBroker is not co-located with it, because in WS-BrokeredNotification the publisher initiates the relationship, and hence controls where it sends its messages. Of course, both the broker and the publisher are also NotificationProducers, and still have to defend against malicious subscriptions using their underlying WS-BaseNotification functionality (see Section 3.3.2.5).

3.3.4 WS-Topics

The WS-Topics specification describes a mechanism for unambiguous definition of "Topic Spaces": collections of topics with hierarchical (parent-child) relationships, and optional equivalence relationships (known in WS-Topics as aliases). Topics can then be associated with notification messages, and WS-Topics used to infer how these messages relate to other topics in the hierarchy.

WS-Topics thus provides a standard way to manage the distribution of notification messages, which is most useful in WS-BrokeredNotification where publishers can explicitly register the topics on which they will produce notifications.

3.3.4.1 Adoption Considerations

This specification does not pose any security risks. It simply provides a means to define a topic hierarchy that can be used when interpreting the scope of subscription or publisher registration requests. There is no need to constrain or profile the use of WS-Topics as a notification topic schema.

3.4 OGSA Profiles

3.4.1 OGSA WSRF Basic Profile

The OGSA WSRF Basic Profile 1.0 mandates that endpoint references to WS-Resources must conform to the WS-Addressing 1.0 specification. It then defines four resource properties that must be provided to support a level of introspection:

- ResourcePropertyNames: giving the element Qnames of all available resource properties;
- FinalWSResourceInterface: giving the “final” portType supported by the WS-Resource;
- WSResourceInterfaces: giving a list of all known portTypes supported by the WS-Resource;
- ResourceEndpointReference: giving the endpoint reference for the WS-Resource.

The profile goes on to mandate that two optional features of WS-ResourceProperties also be supported:

- the GetMultipleResourceProperties method must be supported, for access to ResourcePropertyNames;
- the QueryResourceProperties method should be supported, with the X-Path 1.0 query language.

It then asserts that “OGSA services fundamentally rely on the ability to understand changes in other OGSA services”, and that ResourcePropertyValueChangeNotification be supported, through the use of the WS-BaseNotification producer functionality. It isn’t clear that this really is necessary, unless one is using WS-ServiceGroup, which is NOT part of the profile (see below).

Resource lifetime is then covered: OGSA WSRF Basic Profile requires support for both immediate and scheduled destruction as specified by WS-ResourceLifetime. Next, the use of WS-BaseNotification is constrained by requiring that the “UseRaw” element must not be used in the Subscribe request – thus forcing all WS-BaseNotification to be carried via a Web Service message using the Notify message schema.

There are some constraints on WS-BaseFaults, specifically that any extension must not introduce any new child elements to the sequence in the BaseFaultType element, so as not to violate the Unique Particle Attribution constraint.

There is no reference to WS-ServiceGroup in the OGSA WSRF Basic Profile. It should be recalled that WS-I Basic Profile mandates UDDI as the basis for a service registry implementation. OGSA WSRF Basic Profile aims to extend the WS-I Basic Profile, and this may be why WS-ServiceGroup has been left out.

Finally, the OGSA WSRF Basic Profile mandates that in addition to its conformance criteria, any implementation must also conform to an OGSA Basic Security Profile.

3.4.2 OGSA Basic Security Profile

The OGSA Basic Security Profile builds on the WS-I Basic Security Profile 1.0 (for using transport and message-level security) and the WS-I SAML Token Profile 1.0. The OGSA Basic Security Profile comes in two flavours:

- the Anonymous Channel profile for use in “safe” environments such as secure LANs; and
- the Secure Channel profile for use in “unsafe” environments such as the Intranet.

The only significant difference between the two is that the Anonymous Channel does not require mutual authentication between message senders and recipients, on the basis that this could be provided safely by other means in a LAN or Intragrid environment. We therefore focus on the Secure Channel profile, while noting that in some circumstances the Anonymous Channel profile may be relevant.

The profile specifies the following conformance criteria:

- transport security (HTTP or non-HTTP transport using TLS) is mandatory, as defined in the WS-I Basic Security Profile, and must use mutual authentication;
- message-level security is optional, but if used must include both signature and encryption as defined in the WS-I Basic Security Profile, and incorporate an X509 certificate (or other authentication token) in the security header;
- any endpoint using message-level security must include its encryption key within a meta-data element in its WS-Addressing Endpoint reference;
- other assertions may be made by a message sender by attaching security tokens to the message, which must be either X509 attribute certificates or SAML tokens.

Nothing in the Basic Security Profile discusses authorisation, except to note that authorisation depends on authenticated identity and other assertions conveyed in security tokens.

3.4.3 Adoption Considerations

WS-Notification depends on the adoption policy for WS-ResourceProperties. OGSA WSRF Basic Profile requires exposure of resource properties via GetMultipleResourceProperties as well as GetResourceProperty methods, and recommends that QueryResourceProperties with XPath 1.0 queries should be supported as well. To do this safely would require a security policy and enforcement mechanism that recognises the semantics of resource properties and query language.

The introspection properties required by the Basic Profile do not cause any new problems in themselves. If the resource properties are read only, then they will be the same for all WS-Resources accessible via a given service, and so cannot be considered security critical. If a user has any access to a WS-Resource, it is reasonable that they can read these introspection properties. This could be enforced, but only by adopting the “per property” security policy.

OGSA Basic Security Profile does not consider access policies for resource properties and notification messages. It confines itself to defining how transport and message level security should be implemented, and does not consider what access policy should apply to resource properties or notification messages mandated by the OGSA WSRF Basic Profile.

OGSA Basic Security Profile also defines a rather “non-standard” mechanism for key exchange, encoding keys into EPR, rather than using profiles on existing specifications such as WS-Policy, WS-Trust or WS-SecureConversation. The apparent requirement to use encryption and signature together or not at all in message-level security is also odd, since the profile mandates authenticated transport-level security, and encryption at both transport- and message level represents a significant overhead that is only needed if transporting confidential data via an intermediary.

4 Industrial Grid Profile Recommendations

4.1 Purpose

From the analysis of the Grid-related specifications, it is clear that for industrial application requirements there are many adoption considerations. However, some aspects of the specifications are relevant to industrial needs and it is worth adopting these aspects where we can do so safely. SIMDAT can then demonstrate a commitment to making the technology accessible to the wider community, and make it more likely that these communities will adopt SIMDAT results. Clearly, SIMDAT should only adopt these specifications when to do so would not unduly compromise project objectives.

The recommendation is to design an Industrial Grid Profile to meet the security and operational requirements that are important to industry, and implement it in successive SIMDAT developments. SIMDAT will publish profiles with each of its solution portfolios, along with associated white papers justifying the choices made when defining them.

The initial industrial profile is based on three levels of conformance recommendations taking into account the maturity and understanding of how the specifications can be adopted:

1. Basic WSRF conformance;
2. Wider WSRF conformance;
3. OGSA WSRF Profile conformance.

The following recommendations address each of these aspects in more detail. It should be noted that the specifications are changing rapidly, as OGSA develops its architectural vision and that we will continue to revise the conformance analysis considering amendments to specifications and lessons learnt through industrial deployments.

4.2 Basic WSRF conformance recommendations

It is recommended that SIMDAT SHOULD adopt WS-Addressing, using EPR to encapsulate conversation IDs. We should avoiding using ReplyTo and FaultTo headers, and impose a white list defining EPR reference parameters that are understood and can be elevated to full headers in messages addressed using EPR, and on the HTTP arguments included in EPR Address elements.

It is recommended that SIMDAT conforms to the WSRF specification, as follows:

a. Adopt the WS-Resource specification, but choosing a single context identifier element (e.g. ResourceID) for all services, and retaining a minimal white list containing only this chosen identifier element when processing WS-Addressing EPR.

b. Implement the GetResourceProperty operation only from WS-ResourceProperties, but declaring no resource properties in the first instance. To ensure this operation cannot be used to probe for meaningful ResourceID, the security policy should deny access to this operation for all users who don't have other access rights to the same ResourceID.

c. Define a BaseFault to be returned whenever a user attempts to access a contextualised service using a ResourceID for which they are not authorised or which does not exist. Using the same fault response in both situations ensures no disclosure of information about resource identifiers.

d. Convert all other Faults into BaseFaults

4.3 Extended WSRF/WSN conformance recommendations

It is recommended that WS-ResourceProperties MAY be used, but that resource properties must be a fixed set of “well-known” properties accessible published by the service provider and updated only to clients and services hosted by the same service provider as a WS-Resource. This limits possible interoperability problems from the un-encapsulated use of resource properties, and the need for complex security policies and associated policy enforcement and management mechanisms.

It is recommended that WS-ResourceLifetime SHOULD be adopted, subject to security restrictions on direct access to its resource properties. This is an optional part of WSRF, but is mandatory under the OGSA WSRF Basic Profile.

It is recommended that the WS-ServiceGroup specification SHOULD NOT be adopted. It is not part of the OGSA WSRF Basic Profile, and requires the implementation of a complex security policy infrastructure that does not exist today. These drawbacks suggest that WS-ServiceGroup, although one of the first WSRF specifications, may be subject to significant changes or even deprecation in future.

It is recommended that WS-BaseNotification, WS-BrokeredNotification and WS-Topics SHOULD be used as follows:

- ➢ all notification messages should use the Notify message format; and
- ➢ subscription should be accessible only to services and clients hosted by the same service provider as a NotificationProducer; and
- ➢ publisher registration should only be allowed for publishers that are hosted by the same service provider as a NotificationBroker.

The co-location restrictions eliminate the possibility of a notification producer being used by a malicious third-party subscriber for denial of service attacks or to leak data, but without requiring complex security policy and associated policy enforcement and policy management mechanisms.

4.4 Full OGSA WSRF Basic Profile conformance recommendations

To fully conform to the OGSA WSRF Basic Profile is difficult, because it forces SIMDAT to support the full WS-ResourceProperties functionality and WS-BaseNotification. This means we would have to solve the security issues associated with the corresponding operations, as well as implementing the rather complex WS-BaseNotification components and workflows.

There are some obvious questions we should analyse in detail before conformance recommendations can be made:

- ➢ If we deny all access to all WS-ResourceProperties operations, or restrict access only to the service provider of each WS-Resource, which features of the OGSA WSRF Basic Profile can be implemented and made useful, if any?

- If we restrict access to GetResourceProperty only, then a single Qname would be the query argument in each request, and the set of available resource properties will be fixed. Could we support authorisation policies easily in that case? If so, how much OGSA functionality could be implemented securely with this restriction in place?

- Can we identify a fixed set of non-sensitive resource properties and notification topics that is sufficient to implement OGSA? If so, does it make sense to specify a profile that only uses these? Does this make sense if we assume that any other resource properties or topics are inaccessible to any but the service provider?

- If we allow free access to all WS-ResourceProperties operations, which OGSA WSRF Basic Profile features would become dangerous or unacceptable to industrial users?

- Given any reasonable security policy with respect to resource properties, can we infer the security policy that should apply to WSN components and topics required by OGSA?

What is clear is that we should not adopt OGSA WSRF Basic Profile at present. To do so may be possible following wider testing and discussion of OGSA WSRF Basic Profile, leading to consensus on how to handle the security concerns through changes to the profile or identification of appropriate and realisable security policies for OGSA functionality.

5 Conclusions

In this document we have presented an analysis of key emerging Grid-related specifications and adoption considerations in the context of industrial, B2B applications. We have described a strategy for incrementally defining an Industrial Grid Profile that constrains the specifications, so that they can be safely adopted by industry considering security requirements, operational requirements and the stability of current drafts. The publication of Industrial Grid Profiles will enable SIMDAT to engage and influence the future development of these specifications.

The first Industrial Grid Profile to be implemented and justified was the basic WSRF profile described in Section 4.2. This profile has been implemented in GRIA 5 and has been validated by SIMDAT application activities during the Interoperability Phase of the project.

The second Industrial Grid Profile to be implemented would be the Extended Profile described in Section 4.3. This is probably the first for which we could make a reasonable comparison with the OGSA profile, though the analysis would have to be revisited with reference to the then current OGSA draft and changes as a result of WS-Convergence. The concern with OGSA at this stage is clearly the security policy and enforcement challenge of making a very wide range of properties accessible through a single mechanism in an industrial context. We would have to fully understand this security problem before making recommendations to industry and the OGSA working group.

Clearly, the specifications will carry on changing rapidly, as OGSA develops its architectural vision. We will continue to monitor and analysis these amendments feeding back to the SIMDAT consortium on adoption options for industrial Grid deployments.