Abstract—Web Services are the technology of choice for Internet-based applications with loosely coupled clients and servers. That made them the natural choice for building the next generation of grid based applications.

However, Web Services do have certain limitations. The Web services architecture lacks in providing the notion of state, stateful interactions and notification of state changes which are the main requirements of dynamic virtual organizations. As these issues are of central concern to the developers of distributed systems, statefulness is considered an important concept and several standards, specifications and implementation projects have been presented for this purpose.

In this paper we discuss the concept of stateful Web services and present a review of the technologies, projects and initiatives which are provided to support statefullness for Web services and also present a comparison between different existing specifications and their related implementations.

Index Terms—OGSA, OGSI, Grid Services, WSRF, WS-Notification, WS-Addressing.

I. INTRODUCTION

Even though there is nothing in the Web Services architecture that prevents it from being stateful, plain Web services are stateless. This means that the Web services can't "remember" information, or keep state, from one invocation to another and after an invocation the following invocations have no idea of what was done in the previous ones. In contrast, Stateful Web Services can receive a series of requests from a particular client and remember some state information between requests; in other words they can persist their state between invocations. The distributed computing and specially Grid needed the concept of "stateful resources" which could be discovered, queried, and manipulated via web services. For different reasons, one was the stateless nature of web services, the Grid architects decided that there were enough requirements in their service oriented architecture that were not met by the Web Services standards.

To satisfy these requirements an Open Grid Service Architecture (OGSA) was proposed which was built on concepts and technologies from both the Grid and Web Services communities.

OGSA defined something really challenging: Grid Service, a Web service that conforms to a set of conventions for purposes such as service lifetime management, inspection, and notification of service state changes [1]. These conventions made controlled management of the distributed and often long lived state- that is commonly required in distributed applications possible. The Open Grid Services Architecture described a physiology which was later provided with a standard specification, the Open Grid Services Infrastructure, OGSI.

OGSI defined a set of conventions and extensions on the use of Web Service Definition Language (WSDL) and XML Schema to provide Web Services with statefulness [2]. From then the earliest discussions on Stateful Web Services began.

The fact that Web services don't keep state information is not necessarily considered a bad thing. There are plenty of applications which have no need whatsoever for statefulness. For example, the Weather Web service is a real working Web service which has no need to know what has happened in the previous invocations. However, Grid applications generally require statefulness. So, we would ideally like our Web service to somehow keep state information.

The following parts of the paper presents information on different existing projects on Stateful Web Services, their related technologies and different implementations.

II. OGSI WORKING GROUP, WHERE THE FIRST STANDARD STATEFUL WEB SERVICES COME FROM!

The purpose of the OGSI Working Group was to review and refine the Grid Service Specification and other documents that derive from this specification, including OGSA infrastructure-related technical specifications and supporting informational documents.

The OGSI Version 1.0 specification was completed in June 2003 and released in July 2003. It was built on both Grid and Web services technologies, and defined mechanisms for creating, managing, and exchanging information among Grid services [2].

One of the principal objectives of the OGSI specification was the identification and description of the state, called 'statefulness', to manage interactions between systems and their users. OGSI provided conventions, behaviors and other mechanisms associated with Grid service creation and discovery, controlled, fault-resilient, and secure management
of the distributed and often long-lived state that is commonly required in advanced distributed applications.

This specification was implemented by different projects and on different platforms: The Globus Project implemented OGSI under GT3 (Globus Toolkit 3) project in java. Later there was another implementation of OGSI on .NET platform by the name of the OGSI.net project, a work of the University of Virginia Grid Computing Group [1]. OGSI::Lite was also an experiment in creating Grid Services using Perl.

OGSI was the first standard specification which implemented statefulness on web services and many projects benefited from its useful capabilities but it didn’t take long since another standard with more noticeable characteristics took its place.

### III. WSRF – Web Services Resource Framework

The question that first comes in mind is why it seemed necessary to migrate to a new standard specification by the name of WSRF. Here is the answer:

In parallel with and subsequent to OGSI work, the Web services architecture evolved for example with the definition of WSDL 2.0 and the release of new specifications such as WS-Addressing and WS-Notification. These developments made it timely to consider how the functional capabilities of OGSI can benefit from the functionality provided by other specifications and to align OGSI functions with the emerging consensus on Web services architecture.

Besides OGSI 1.0 combined functions that were independently useful, for example event notification, into one specification. It was appropriate to factor the OGSI interfaces to produce a framework of independently useful Web service standards [2].

On the other hand, OGSI modeled a stateful resource as a Web service that encapsulates the resource’s state, with the identity and lifecycle of the service and state coupled. This was a serious concern to the Web services pundits who argued Web services should be stateless.

Finally, due to aggressive use of XML Schema (e.g., xsd: any attributes) and document-oriented WSDL operations, the specification did not work well with existing Web services and XML tooling.

For the reasons above in January 2004, the WS-Resource Framework was proposed as a refactoring and evolution of OGSI and aimed at exploiting new Web services standards, specifically WS-Addressing and WS-Notification, and at evolving OGSI based on early implementation and application experiences [2]. The new specification covered all the problems and shortcomings of the previous one:

Instead of putting the state in the web service, which is generally regarded as a bad thing and thus make it stateful, WSRF keeps it in a separate entity called a resource, which will store all the state information. The composition of the Web service and the stateful resource is then called a WS-Resource [3].


In April 2006 these specifications of WSRF v1.2 were accepted as an OASIS standard. Here we present a review of the specifications:

- The WS-ResourceLifeTime provides mechanisms for WS-Resource destruction, including message exchanges that allow a requestor to destroy a WS-Resource, either immediately or by using a time-based scheduled resource termination mechanism [5].
- The WS-ResourceProperties specification provides mechanisms for definition of a WS-Resource, and also mechanisms for retrieving, changing, and deleting WSResource properties [6].
- The WS-ServiceGroup provides an interface to heterogeneous by-reference collections of Web services [7].
- And WS-BaseFaults provides a base fault XML type for use when returning faults in Web services message exchange [8].

While WSRF is important particularly for grids, another important feature of it is that it addresses fundamental architectural issue in doing Object Oriented like approach on web services.

WSRF exploits recent developments in Web services architecture including WS-Addressing and WS-Notification. As these two specifications have strong ties to WSRF, they have been considered in this paper.

### IV. Web Services Notification-based Projects

According to IBM the Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. As a result the notification model, where a service provider or other entity, initiates messages based on a subscription or registration of interest from a service requestor, was a common pattern that was to be standardized in Web services [9]. This notification pattern is increasingly being used in a Web services context.

Entities interested in receiving information which are called Notification Consumers can be registered dynamically with the Web services which are able to produce the information – called Notification Producers. Dynamically registration means that Notification Producer needs to have no knowledge of a Notification Consumer prior to its registration. When registering, a Notification Consumer can inform the Notification Producer of the nature of the information it is interested in.

Several Event-driven, or Notification-based, projects for web services have been presented; in the following we present a review of the most important ones:

#### A. WS-Events

The WS-Events component was published in July 2003 by HP in a Web Services Management Framework and defined the Web services based on event notification mechanism. It defined an event - a change in the state of a resource or...
request for processing. Event producers, event consumers and event brokers.

An event producer is an entity which generates notifications; an event consumer is the receiver of notifications and an event broker is an entity which routes notifications.

B. WS-Eventing

In January 2004 a draft version of Web Services Eventing (WS-Eventing) was published. The WS-Eventing specification described a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages.

Supporting both SOAP 1.1 and SOAP 1.2 Envelopes, the specification allowed subscriptions to be renewed and defined expiration for them.

C. Web Services Notification Framework

A Web Services Notification Framework was announced on March 05, 2004. The goals of WS-Notification were to standardize the terminology, concepts and message exchanges the WSDL needed to express the notification pattern. It was also intended to provide a language to describe Topics and was supposed to be a framework independent of binding level details, which would allow for Message Oriented Middleware implementations, for federation of brokers, simple transformation and aggregation of Topics, and was supposed to be composable with other WS-* specifications.

WS-Notification is divided into three specifications:

1) WS-Topics: which are used by the other two specifications in WS-Notifications to present a set of items of interest for subscription[10].

2) WS-BaseNotification: in which Notification producers have to expose a subscribe operation that notification consumers can use to request a subscription. Consumers, in turn, have to expose a notify operation that producers can use to deliver the notification[11].

3) WS-Brokered Notification: which states that notifications are delivered from the producer to the consumer through an intermediate entity called the broker[12].

On October 11th 2006, OASIS announced that its members had approved the WS-BaseNotification, WS-Brokered Notification and WS-Topics specifications as OASIS standards.

The WS-Notifications family of specifications like WSAddressing (which is described later) is not a part of WSRF, but has strong relation with it. A publish/subscribe notification mechanism (WSNotification) can be built on top of the WS-Resource Framework by creating subscriptions to state changes within the WS-Resource to monitor and report on resource property changes[4].

VI. WS-ADDRESSING

WS-Addressing provides a mechanism to address web services and messages which is much more versatile than plain URIs and provides transport-neutral mechanisms for this goal.

Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages.

It also enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

WS-Addressing introduced two new constructs for the web services vocabulary: Endpoint-references and message addressing properties.

1) Endpoint is an established term for a destination at which a web service can be accessed. Endpoint references are defined as a complex type in the WS-Addressing schema. The endpoint reference type contains an address (a URI), reference properties, reference parameters, a port type, a service name, and policy elements (defined by the WS-Policy specification). The only required element of an endpoint reference is the address, so the simplest possible endpoint reference is essentially a URI.

2) Message addressing properties, which may include one or more endpoint references, provide a context for that destination information.

WSRF applies WS-Addressing endpoint references to address and access WS-Resources.

VI. GRID SERVICES VS. WEB SERVICES

Since Open Grid Services Infrastructure specification defined Grid services as an extension of Web services (by introducing statefulness and transience) people often try to understand the relationship between Grid and Web services or compare them. The differences between Grid and Web Services were always considered an important and discussible question.

Although WSRF has moved us to a new point, in which we should ask “Should we compare Web and Grid Services?” instead of asking “What are the differences between Web and Grid Services?”, we’ve presented major differences of these two services:

While Web services are stateless and non-transient - for they do not have the concept of service creation and destruction and thus managing their lifecycle- Grid Services solved the stateless and non-transient problems of Web services by introducing a factory/instance model. Instead of having stateless services shared by many clients, a Grid service factory is used to create and maintain multiple instances of the Grid service, each representing one resource.

It is also possible to have one-to-many, many-to-one, and many-to-many interactions between clients and instances of Grid services which are transient and are being bound to the lifetime of the Grid service’s container.

Grid service can also be configured to be a notification source and some clients can be configured to be notification sinks (subscribers) so that if a change occurs in the Grid Service all the subscribers will be notified of that.
However, despite Grid services addressed problems which were related to statefulness, transience, lifetime management and state change notifications, they were not widely adopted by the Web services community and questions were raised about the significance of Grid services. For this reason and the ones entitled at the beginnings of part III WSRF has been adopted as the basis for Open Grid Services Architecture based Grids. Subsequently, the term "Grid service" was deprecated by the OGSA, and as a consequence, from the OGSA perspective the term has no technical significance and its usage is discouraged [13].

VII. WSRF IMPLEMENTATIONS – DISCUSSIONS AND COMPARISONS

There are currently five implementations for WSRF standard which are presented by different teams and different implementation languages, programming modals and overall goals.

These implementations include:

1) GT4-JAVA : the Java Web Service Core of the GLOBUS Toolkit 4.
2) GT4-C : the C Web Service Core of the GLOBUS Toolkit4.
3) pyGridWare: a python WSRF implementation which is also distributed with GT4 as its Python Web Services Core.
4) WSRF::LITE: the perl based WSRF.
5) WSRF.net: an implementation of the WSRF and WS-Notification on .net framework.

In [14] these different implementations of WSRF specifications from the perspectives of architecture, functionality, standards compliance, performance, and interoperability are compared in detail.

In this paper we briefly mention some of the major differences.

- For processing the HTTP protocol messages used to transport requests and responses, and for deserializing and serializing the SOAP messages, Apache Axis is used in the case of GT4-Java, Zolera SOAP Infrastructure (ZSI) for pyGridware, SOAP::Lite for WSRF::Lite, and Microsoft Internet Information Services (IIS) and the Web Services Enhancements (WSE) for WSRF.NET. The implementations are different from each other in robustness and performance but they offer similar capabilities.

- While a SOAP request is received and security processing are done, the relevant operation must be identified and dispatched. Then, any reply is constructed and returned for SOAP processing and transport. In each system WS-Resources exist inside a container process within which these operations take place. WSRF.NET uses ASP.NET, GT4-Java can use Tomcat/Axis, and pyGridWare uses Twisted. Some systems provide their own containers: GT4-Java and pyGridWare can run stand-alone, GT4-C has its own container, and WSRF::Lite provides two containers.

- Once a WS-Resource is created it should persist its state between invocations. This can be achieved by holding the resource in memory, writing it to disk, or storing it in a database. All the five specifications except WSRF.net persist WS-Resources in memory by default which gives better performance in response time, but has lower fault tolerance. WSRF.net persist WS-Resources in database. This approach is slower than in-memory storage (although write-through caching makes it competitive), but provides fault-tolerance and access to powerful query/discover mechanisms that do not exist in the file system approach.

- In accessing WS-Resource state, GT4-Java and pyGridWare implement a Resource Home interface which contains the find() method and discovers resources based on a supplied key, such as the resource name, while WSRF.net uses a database query. Queries can not only lookup resources based on unique key values, but also can find them based on the data contained within the resource. GT4-C can define interfaces for manipulating resources and provides a default implementation of those interfaces. In case of WSRF::Lite the state can be implicitly in the services execution context, or the Container can provide a key to the developer which is used to find the state.

- In providing WS-Notification, WSRF.NET has implemented all the three WS-Notification specifications described. GT4-Java and pyGridWare do not implement WS-Brokered Notification and only support flat topic spaces and basic subscriptions. GT4-C does not implement producer-side notification (Notification Producer, Subscription Manager) while WSRF::Lite does not support any Notification specifications.

VIII. CONCLUSION

Stateful, transient Web Services are important requirements of the Grid environment and different evolving standards have been provided for this reason. This paper has put forward an approach to provide concepts, evolving standards, projects and their different implementations on stateful, transient Web Services. We have also presented the evolution of these standards and the recently accepted ones. A brief comparison of various implementations of the new standards have also been provided which in our case proves the importance and usefulness of Stateful Web Services.

REFERENCES


Copyright of International MultiConference of Engineers & Computer Scientists is the property of International Association of Engineers and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.