School of Computing Science,
University of Newcastle upon Tyne

Grid Computing Using Web Services

Savas Parastatidis, Paul Watson, and Jim Webber

Technical Report Series

CS-TR-926

August 2005

Published by the University of Newcastle upon Tyne,
School of Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, UK.

Grid Computing using Web Services Technologies

Savas Parastatidis, Paul Watson, Jim Webber

School of Computing Science
University of Newcastle upon Tyne, UK
{Savas.Parastatidis, Paul.Watson, Jim.Webber}@newcastle.ac.uk

Abstract. Service-Oriented Architecture (SOA) is the contemporary paradigm of choice for developing scalable, loosely-coupled applications that span organisations. However the architectural paradigm that is SOA is often confused with the implementation technology that is Web Services. In this paper we aim to clarify the fundamental tenets of SOA and their relevance to Internet-scale computing (or Grid computing). We then show how to apply the principles of SOA to building Internet-scale applications using Web Services technologies and how to avoid software pitfalls by adhering to a number of deliberately simple architectural constraints.

1 Introduction

With the advent and subsequent rise to prominence of Web Services, there has been renewed enthusiasm for service-orientation and Service Oriented Architectures in the development community. While service-orientation is independent of, and pre-dates Web Services technology, the rise and rise of Web Services has meant that the application of SOA has become de rigueur for architects and developers.

Concurrently, 'Grid computing' [11] has emerged as a popular paradigm for enabling the formation of virtual organisations and for integrating distributed resources. A significant investment in terms of capital and human resources has been made in architecting the vision of Grid computing around the concepts of service-orientation [10] using Web Services technologies as an implementation technology.

However there is common misconception concerning Web Services technologies in which they are seen as a form of software magic which automatically yields a loosely coupled solution which is scalable, robust, and dependable. There is sometimes the assumption that the use of Web Services technologies is sufficient to implement high quality Grid applications. It is certainly possible, and generally desirable, to build Grid applications using Web Services protocols and toolkits. However it is equally possible to build such applications in ways that violate every architectural principle and tenet of SOA and lack the characteristics of SOA-based systems.

The central tenet of this paper is that Grid applications built using Web Services technologies maximise their potential only when implemented in a manner that follows the principles of SOA, as opposed to alternative approaches such as platform-independent RPC or distributed Object-Orientation [29].

The views we present here are based on a distillation of implementation effort and experience [22, 24, 25, 35], and form the basis of a simple and scalable abstract view of service-orientation upon which real concrete Grid applications can be based. The rest of this paper is structured as follows: Section 2 briefly introduces the term "Grid computing" and puts it in the context of this paper. Section 3 discusses Service Oriented Architectures independently of any implementation technology, while Section 4 describes how the suite of Web Services technologies could be used to implement service-oriented applications. Section 5 presents a set of principles for building Web Services-based applications while Section 6 discusses the relationship of the suite of Web Services protocols with the principles for Service-Oriented Architectures. Finally, Section 7 draws conclusions.

2 Grid Computing

The vision of Grid computing has evolved from interconnected supercomputers in the 1990s, to a paradigm for Internet-scale, inter-organisation computing. While there is no widely-accepted definition of the term 'Grid computing,' some common uses are:

- 'Utility computing' which is about providing computing resources (e.g. CPU, data storage, access to specialised devices, etc.) in a seamless fashion to end users similarly to the way electricity is delivered to our homes (e.g. [14]).
- 'On-demand computing' which is a term usually used by vendors to promote the concept of outsourced computing and enabling services (e.g. [15]).
- 'Seamless computing", or the interconnection of computing facilities and transparent access to computational, data, and other resources and services (e.g. [20]).
- 'Global data integration' where information is allowed to flow between organisations after the necessary security, trust, policy, privacy, etc. restrictions have been put in place (e.g. [23]).
- 'SETI@home' [3] type applications where communities of altruistic individuals are formed to solve large computational problems (e.g. [7]).
- 'Virtual organisations', or the infrastructure necessary for the dynamic formation, management, and exploitation of alliances between organisations in order to achieve a common goal (e.g. [12]).
- 'Universal computer' where the Internet becomes the operating platform for all users' applications (e.g. [8]).

Irrespective of which of the definitions is adopted, it is clear that those working on building the Grid computing vision have a large set of interesting problems to address, like the pooling of computational capacity, data integration, security, digital contracts, service-level agreements, negotiation, policies, quality-of-service, dependability, electronic payment, etc. Such problems, often encountered in distributed systems research, now have to be addressed and applied at magnitudes up to and including Internet scale. It is due to its large scale and the common belief that service-orientation is the most appropriate paradigm for addressing these issues that

we define 'Grid computing' as *Internet-scale, service-oriented computing* and choose to make use of Web Services technologies to provide the underlying infrastructure.

We argue that Grid system architects face a similar set of problems whether they apply the vision of Internet-scale, service-oriented computing within or across organisation boundaries. We argue that the same set of solutions can be applied in both cases.

Inside Organisations

Within organisations the notion of sharing computational resources, data, network and so forth is already an established practice. However to-date that practice has occurred on a per-enterprise basis where application and data integration is managed at the enterprise architecture level. While enterprise architecture is invaluable in managing today's IT infrastructure because of the proprietary nature of a typical rollout it is difficult to transfer anything other than best practices between projects.

The promise of Grid computing at this level is primarily the opportunity for virtualising access to computational and data resources in a standardised fashion. That is, to make access to typical enterprise resources seamless and repeatable between the different entities of an organisation.

Across Organisations

When working across organisations there are new challenges for Grid computing, different from the kinds of problems faced when working within a single administrative domain. At this level the Grid addresses Internet-scale computing issues including federation of identities, contracts, service-level agreements, quality of service, etc.

The promise of Grid computing at this level is that it will provide a suitably constrained architecture and framework for Internet computing. That is, it will allow applications to be built and integrated with other arbitrary applications exposed across the Internet whilst maintaining high levels of quality of service and be resilient to increases in workload and robust in the presence of failures. As a consequence, new types of science and commercial applications and services will emerge.

3 Service-Oriented Architecture

Service Oriented Architectures [2, 13, 18, 28, 29] exist independently of any specific implementation technology like Web Services, but it was the advent of Web Services, and its accompanying hype, which reinvigorated interest in service-orientation and SOA.

However as researchers and developers have shifted their work to be in vogue with the latest buzzwords, the term SOA has become overloaded. Therefore before discussing how to build Web Services applications, the fundamental constituents of SOA must be pared from the hyperbole surrounding it. In this section we present an

abstract view of Service-Oriented Architecture, which we will later concretise in terms of Web Services.

During the course of our work on Web Services and Grid computing [24-26], we have identified what we believe are the two fundamental components of SOA upon which all higher-level functionality is built:[1]

1. **Services.** A service is the logical manifestation of some physical or logical resources (like databases, programs, devices, humans, etc.) and/or some application logic that is exposed to the network that may be executed in response to the arrival of messages.
2. **Messages.** A message is a unit of communication for exchanging information. All communication between services is facilitated by the sending and receiving of messages.

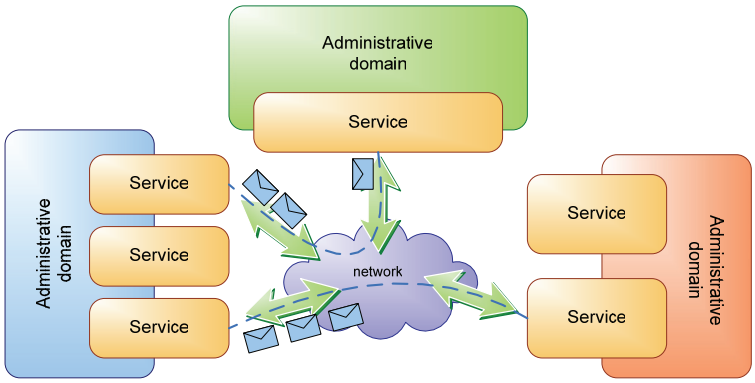

Fig. 1. The relationship between services and messages

Fundamentally Service-oriented systems are based on message-passing not on higher level abstractions like method calls[2]. It is this characteristic which enables loose-coupling since it allows services to be created and versioned in isolation based on message-level contracts. Services are not permitted to share knowledge of the internals of other services, but only exchange messages with them within the context of specific applications, as shown in Figure 1.

Given the importance of the two fundamental building blocks of SOA, in the following sections we explore the makeup of services and messages.

[1] Note that the W3C's Web Services Architecture document [33] presents a total of 16 components in its service-oriented model, plus a large number of interrelationships. This is not at odds with our view since it is a higher level view of the architecture.

[2] Method calls and events are often useful abstractions at the application level and most Web Services toolkits build such abstractions on top of the network-level messaging libraries. This can improve developer productivity but can be dangerous if developers fail to understand that once outside of a service's boundary, the abstractions which are presented to them as method calls and events are actually message exchanges.

The Anatomy of a Service

The architecture of a generic service is shown in Figure 2. Outwardly, a service is simply an addressable endpoint which processes messages. The internal architecture of a service is a classic N-Tier architecture utilising a message-router pattern. The beauty of Service-Oriented Architecture is that it is not revolutionary, but ordinary and therefore comprehensible to any proficient software engineer.

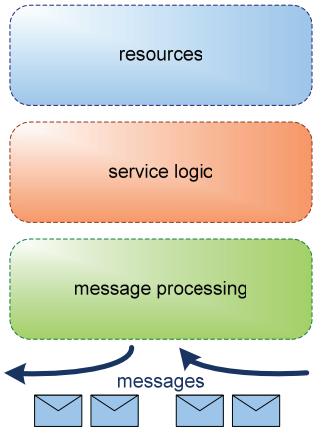

Fig. 2. The anatomy of a Web Service

The arrival of a message at the service endpoint normally causes the message to be validated by the messaging layer (although there may be situations where validation may not occur until further up the stack). Once validated, the message can be internally dispatched up the service stack and ultimately cause some processing at the logic layer of the service.

For ease of recovery and scalability, the service logic layer for an individual service implementation should manipulate only soft state; that is state which can be recomputed or recovered in the event of failure. Using soft state (effectively making the service implementation stateless) means that if a service fails, a backup service can be seamlessly brought online or the service can recover gracefully once the cause of the failure has been rectified. Maintaining only soft state in the service logic is important, since it alleviates the need for services to contain intricate recovery and consistency routines.

The uppermost layer in the stack is the resources, often representing persistent state, which may be shared by many copies of a service, and indeed by many services. This is where the enterprise data resides in a variety of hardened data storage mechanisms like (transactional) databases and queues and sometimes in less hardened media such as card files and human memories[3].

[3] The choice of enterprise storage is important for the architect of an individual service, yet fundamentally out-of-scope for an application which consumes that service.

Service Intercommunication

While understanding the tiered architecture of a service is of paramount importance for service architects, application architects have a different set of concerns. A service-oriented application is an aggregation of services, where the application orchestrates the message exchanges between services in order to facilitate some domain-specific work.

The underlying transport protocol for transferring messages may vary from application to application and may differ between different message exchanges within the same application. Depending on the level of quality of service required from a particular message exchange, an architect might elect to use a reliable message transport for a specific service (such as a queue) or use something more lightweight like TCP/IP. It is however important to distinguish that at this abstract level of architecture the fact that messages are transferred is the key notion, and the details of moving bits over the wire is architecturally transparent.

No matter how messages are ultimately moved across the network, messages themselves are *rich*. Rich messages are self-descriptive and meaningful in the context in which they are sent and received. To be truly meaningful, a message must contain all of the information that a service requires to execute its application logic. This not only reduces network overheads (which may be significant in an application which spans enterprises) but also supports stateless interactions (c.f. HTTP) which improves the prospects for scalability and reliability (as exemplified by the WWW) [9].

However, being meaningful does not necessarily imply any shared understanding beyond the structure of a message; it implies only that both sender and receiver understand the message within their own scopes, orchestrated by some overarching application or business process which understands the overall application or process semantics. That is, services themselves are unaware of the processes which they will support and are therefore able to be integrated with arbitrary partners and business processes.

The use of meaningful messages has ramifications for both service and application architects. For the service architect, fewer, richer, messages simplify the design of the service, while improving prospects for scalability, and simplifying fail-over fault tolerance. For the application architect, fewer, richer message exchanges enhance network performance and reduce the likelihood of transient failures disrupting normal application execution.

4 Applying Service Oriented Architectures to Web Services

Having discussed the Service-Oriented Architecture as a conceptual model, we can now proceed to concretise SOA in terms of Web Services. While a Web Service inherits the generic characteristics of a service, we place an additional constraint on the architecture of a Web Service that all messages exchanged must be in SOAP format. SOAP is the de facto standard message transfer protocol for cross-platform message-level interoperability and is universally supported. Furthermore since SOAP is extensible via its header construct, it has become the protocol of choice for the

higher-level Web Services protocols (security, reliability, transactions and so forth). While some may find it contentious, we believe that it is for the greater good that SOA + SOAP = Web Services, and that anything else (for example C++ objects with WSDL descriptions) is not.

While we constrain Web Services to using SOAP for interoperability reasons, for practical reasons we strongly advocate the addition of a service description to a Web Service to ease composition of services into applications since it describes the contract through which a service is willing to be bound. One obvious candidate for describing Web Services is WSDL [32] which can be used to describe the messages that a Web Service understands, and to a limited extend also describe the message exchange patterns for orchestration purposes.

A more powerful alternative is the SOAP Service Description Language (SSDL) [27] which can describe not only the format and chorography of message exchanges that a Web Service supports but has formal underpinnings which enables automated checking of the protocols that a service supports for deadlocks, consistency and so forth.

In addition to the syntactic aspects of a contract, policies can be used to describe the quality of service characteristics that a service supports. In the Web Services arena, WS-Policy [6] is an extensible framework for describing quality of service aspects of a Web Service, and has already been extended to include specific policy frameworks for security, secure conversations, and reliable messaging.

Adding the SOAP constraint and WSDL or SSDL and WS-Policy descriptions to services concretises the SOA abstract architecture presented in Figure 1 into an application and integration platform as shown in Figure 3.

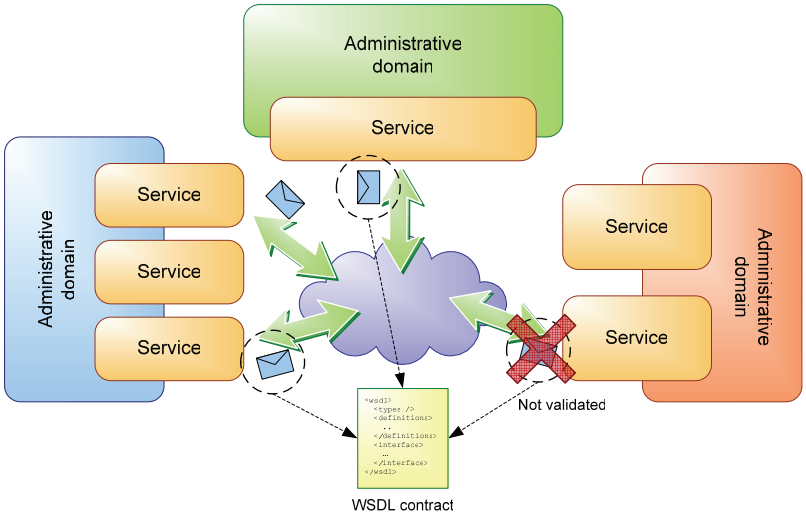

Fig. 3. Message structure and exchange patterns adhere to a WSDL contract

In the following sections we will discuss the basic Web Services model, highlighting salient technologies where appropriate and showing how Web Services

can be constructed and deployed in a manner which is adherent to the principles of service-orientation.

The Anatomy of a Web Service

A Web Service is the logical manifestation of some physical resources and application logic to the network and can be realised as network-capable units of software that implement logic, manage state, communicate via messages, and are governed by policy [21]. Like the abstract service architecture presented above, the canonical Web Service architecture is a multi-tiered artefact built from network, messaging, application, and state layers as shown in Figure 4.

The service logic layer deals only with solving the problem from the application domain. This layer should contain only soft state which, as mentioned earlier, confers benefits in terms of scalability and fail-over fault tolerance. A service containing only soft state typically delegates its requirements for replication and state consistency to the back-end data storage tier (which is designed precisely for such purposes).

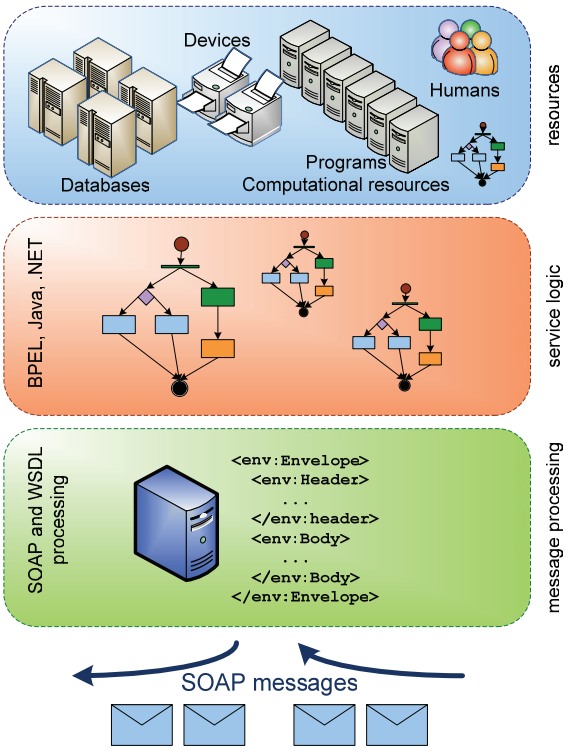

Fig. 4. The canonical architecture of a Web Service

If long-lived state is present within the application layer of a Web Service implementation, these benefits are significantly reduced and the service developer

must implement appropriate failure recovery code as part of the application logic, which is both complex to develop and tends to impact scalability. Without failure recovery code, a newly recovered service would effectively be reset in terms of the conversation it was having with its consumers. The consumers may not expect such behaviour and would most likely fail unless the consumer-service protocol has been designed to allow replays of conversations to occur. Delegating such responsibilities to the consumer (and by implication complicating the service-consumer protocol) is poor practice.

The messaging layer provides the programming abstractions for the application code to exchange messages with other services. The application code has to explicitly reason about those exchanges in terms of messages and message exchanges patterns. The application logic binds to, and directly manipulates message contents, as well as to notifications of the receipt of messages from other services (which may sensibly be delivered via events). In this way, the importance of crossing service boundaries is emphasised and service developers are encouraged to explicitly program services in terms of message interactions and the contents of the messages that make those interactions.

While some Web Services toolkits are beginning to support message-orientation for building Web Services (e.g. WSE [16], Indigo [19], Axis [1]), most toolkits still focus on presenting Web Services as objects. The method call paradigm is flawed in the general case [34] since it does not highlight the difference between invoking a method on a local object and exchanging messages with a remote Web Service. It is clear that the latency and failure modes of a distributed computing environment make distributed computing more complicated than centralised computing, and it is flawed to try to mask the differences [34] – even with Web Services.

Conversely a message-oriented API helps to corral developers into considering the application domain in SOA terms. This in turn loosens coupling since the focus shifts to (validated) messages which, because of extensibility features peppered throughout Web Services technologies (e.g., the introduction of metadata information in a message which can be safely ignored by its ultimate recipients but used by intermediaries to provide a particular quality of service, like security or transactions), can be evolved and versioned over time without breaking existing applications.

The network layer deals with routing of messages to and from the messaging layer. This layer is typically a piece of middleware rather than a component written by the service developer and goes by a variety of designations including the erstwhile “SOAP server”, “SOAP processor”, to the contemporary “Web Services platform” or “Web Services container.” While the network layer is predominantly implemented by off-the-shelf software, it is normal for the layer to be augmented during service deployments in order to expand its capabilities to support non-functional requirements such as security and transactions. Such augmentation is usually accomplished by registering “plugins” or “message processing handlers” from third-party toolkits (or written by the service developer) with the Web Services platform.

This separates the concerns of the functional requirements of the service which are addressed by the service implementation, and the non-functional requirements which are addressed by augmenting the message-processing layer. This decoupling permits the quality of service aspects of a service to evolve independently from the service

implementation and permits different quality of service characteristics to be applied to different service endpoints which share the same implementation.

Given the layering of application, message, and network layers the options for mapping a message-exchange to a back-end action are wide open. While policy descriptions [5] and semantics [30] of an action may augment a service's WSDL or SSDL contract, these should expose intent and not physical service characteristics. We implore service architects to use these layers to their best effect and decouple networking details from application implementation using messages as the interface.

SOAP Messages

In a Web Services environment, we impose the additional architectural constraint that messages are conveyed in SOAP format[4]. That is, for a Web Services-based application SOAP is the transfer mechanism, and in turn it is SOAP messages that are propagated by the underlying transport protocol(s). Whether those protocols are application protocols like HTTP or traditional transport protocols like TCP/IP is unimportant, what is important is that there is a standard model – the SOAP processing model – which provides the fundamental constraints for the entire distributed system architecture.

While in theory any SOAP style is valid, we would advise against using SOAP-RPC (that is rpc/encoded SOAP) because it encourages transmission of application-level objects as parameters to a (remote) procedure call. Instead it is better for the messages that are exchanged to resemble the kinds of business documents that the service's owner deals with. Thus, rather than encoding graphs of objects into SOAP messages, we suggest that un-encoded documents are exchanged (i.e., use of document/literal style SOAP). This advice is underscored by the fact that SOAP-RPC is optional (effectively deprecated) in SOAP 1.2 [31] and the rpc/encoded style is not supported by the WS-I Basic Profile 1.0a [36].

While traditionally SOAP messages do not contain addressing information and instead rely on the addressing of the underlying transport protocol, we add the further constraint that addressing should be part of the SOAP envelope to ensure transport protocol independence. Although there are as yet no open standards for embedding addressing data inside the SOAP envelope, the WS-Addressing [4] specification is an example of one suitable approach (which is fortunately nearing standardisation). In WS-Addressing, the addressing information is placed into a SOAP header block and bound to the addressing mechanism of the underlying transport protocol by the sender of the message. Thusly equipped, SOAP messages can navigate arbitrary networks utilising a variety of protocols for various levels of quality of service and reliability as shown in Figure 5.

[4] While WSDL can supports variety of protocol bindings, we assert that it is SOAP which characterises true Web Services and that it is SOAP which supports interoperability and extensibility that are key to the success of Web Services-based applications.

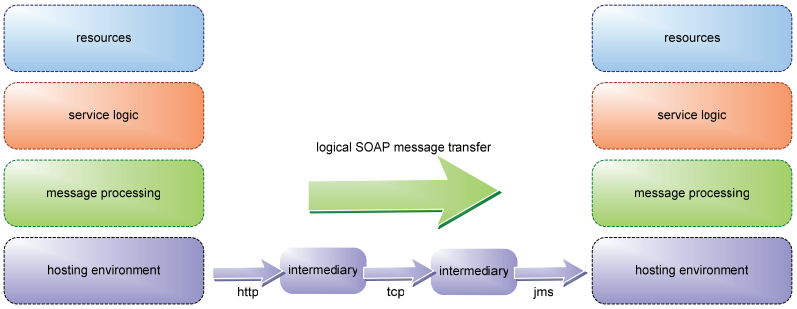

Fig. 5. Embedded addresses enable SOAP transport independence

Upon receipt of a SOAP message the network layer is able to extract information from the header blocks and perform certain processing before the message is delivered up the stack. The information contained in the header blocks may be used, for example, to enlist transaction participants, to authenticate and authorise a message, to decrypt the contents of a message – that is, it provides *context* for the eventual processing of the message. A similar process happens in reverse when a service sends a message, where at the network layer protocol payload can be inserted into the headers, and sections of the body may be re-written according to the rules of the associated protocol – and provide context on the wire for recipients of the message. This is depicted in Figure 6.

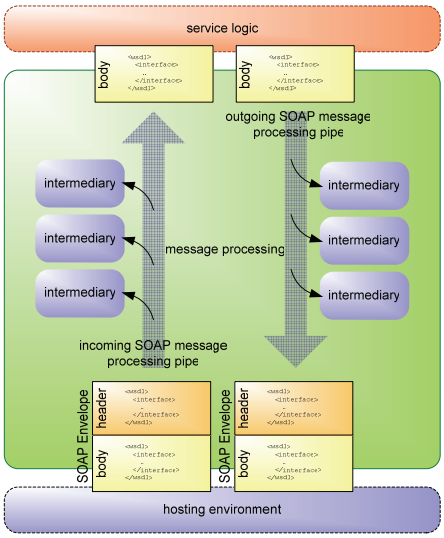

Fig. 6. SOAP extensibility and processing of SOAP messages

The contents of SOAP headers are not fixed, which allows a service to determine its own protocol stack which maps onto the headers in the messages it exchanges. Such extensibility is supported in implementation terms by the plugins registered with the server platform as described above.

5 Architectural Principles for Building Grid Applications with Web Services

Web Services are computational entities which are deployed onto networks of arbitrary scale and as such present additional architectural challenges compared to intranet scale systems. To facilitate the deployment of robust and scalable Web Services-based Grid applications, we suggest the following set of architectural and engineering best practices:

1. Services do not have interfaces in the object-oriented sense, but instead have an associated contract which defines the structure of messages that a service understands, the exchanges into which those messages can be composed, and other policy and quality of service characteristics which the service supports. It is this contract only which defines the externally observable characteristics of the service;
2. Services should be designed not to expose their implementation details or resource representations to consumers. Messages which contain data that directly maps onto a specific implementing object (or that alludes to the existence of such an object) encourage tight coupling and should be avoided. Similarly, mapping of operations at the contract level directly to method names in the service implementation couples implementation and contract and is considered poor practice;
3. Service-based application development should proceed as if the application's developers have no knowledge about the internals of any consumed services, even if they have intricate knowledge in reality. The only understanding a consumer has of the service is its contract through which it advertises supported message exchanges (and possibly policies). Taking such a strict view of service composition supports loose coupling and enables service implementations to evolve without breaking existing applications;
4. The API for service implementations and applications (where the implementation logic meets the network layer) should be cast in terms of the message exchanges that occur. An API which reflects the fact that (potentially) inter-domain message exchanges occur helps to reinforce the notion that services are autonomous and remote and promotes loose coupling.

These rules help to ensure loose coupling since both service and consumer are developed in mutual isolation in accordance to the service's advertised contract. Thus a consumer can choose to use any service which adheres to the same contract, and the service can provision functionality for any consumer which agrees to be bound by that contract. Furthermore, since all the network-level APIs are based on the message-passing paradigm the crossing of boundaries between local implementation and remote service actions is explicit[5].

[5] While some older toolkits may wrap this to appear like objects with method calls (e.g. ASP.Net) the next generation of toolkits expose message-orientation directly to developers (e.g. WSE and Indigo, Axis).

6 Composite Applications and the WS-* Protocols

When aggregating services, especially from multiple administrative domains, into composite applications it is likely that we will require additional quality of service (QoS) features such as security, transactions, and reliable message delivery. Such QoS features are especially needed in the emergent field of Grid computing. Since the set of general principles for service-oriented applications is only concerned with message exchanges which are opaque from an architectural perspective, such quality of service features were not explicitly introduced in the architecture discussion.

However the fact that SOAP supports extensibility through its header mechanism means that quality of service protocol information can be packaged with application messages. Protocol information can be acted on by services to provide such features as non-repudiation, security, encryption, transaction (or activity) scope, and reliable message delivery (Figure 7).

While the WS-* protocols are fundamentally important and a key part of any Web Services toolkit, it is important to understand that they are not part of the model for Service-Oriented Architectures. The underlying architectural principles of services, contracts, and message-passing are pervasive, whereas other protocols are rolled into the stack (and the corresponding SOAP messages) only as needed. A simple parallel is found in the common object-oriented programming languages: object-orientation is a pervasive architectural principle whereas a platform's libraries are included in a program on an as-needed basis.

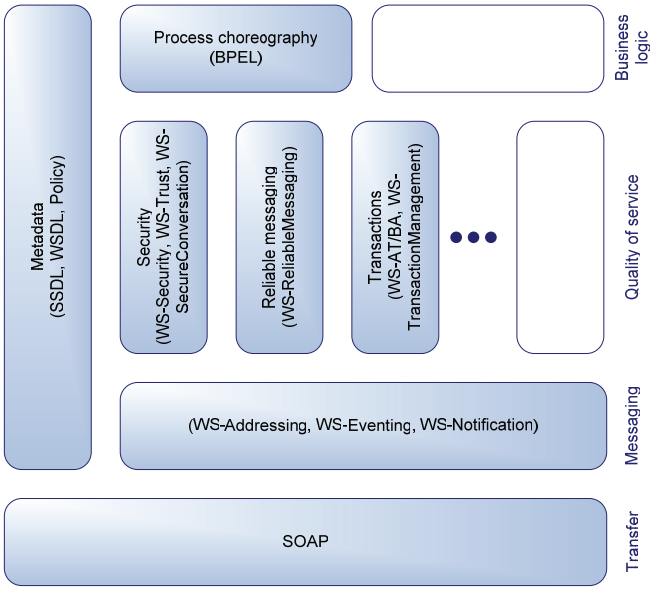

Fig. 7. The Web Services stack (adapted from [17])

However the WS-* protocols are themselves Web Services technology and are therefore not immune to the suggestions made in this paper. Indeed if a piece of

infrastructure violates the principles discussed here, then there is far more scope for havoc than if a single service or application disregards them. The hope is that the developers of the WS-* specifications will maintain their largely good record of creating SOA-friendly, independent, composable protocols, and actively reject any work which does not align with those principles.

7 Conclusions

The terms 'Grid computing', 'service-orientation', and 'Service-Oriented Architecture' have been much overused recently and overloaded with different meanings. Furthermore the suite of technologies that is Web Services has been implicitly linked with SOA leading to false assumption of scalability and robustness of Grid applications simply by dint of the fact that some Web Services technologies (like SOAP and WSDL) have been used.

To counter such misguidance, this paper has presented what we believe to be a concise definition of the abstract SOA and the implied conceptual model based on the notion of services which exchange messages. We have shown that these fundamentals can then be concretised using Web Services technologies to provide the fabric for real world Internet-scale (Grid), service-oriented computing. In particular we advocate a constrained definition of a Web Service to be inline with our interpretation of SOA as a service which exchanges messages in SOAP format. Such services may have an associated contract (in WSDL or SSDL) which describes the format of messages, the message exchanges that service will participate in, and any additional Quality-of-Service features that the service supports.

Deriving from this architecture we proposed a set of simple rules for architecting services, which are designed to keep service implementations loosely coupled and scale to arbitrary size. These rules can be characterised as: Web Services use a message-passing paradigm where contracts govern the message exchanges that a service can participate in, and where no knowledge about the service or consuming application must be assumed or inferred.

We also discussed SOA-friendly mechanisms for aggregating services into applications, with mechanisms for orchestrating message exchanges at the application scope. Finally, we showed how quality of service protocols for "enterprise strength" computing can be layered on top of the architecture.

From these points, we propose that it is possible to support the levels of quality of service that enterprise-grade computing demands (security, reliability, transactions, etc) in a manner that is conformant with the principles of SOA, and thus derives the inherent benefits of that architecture. Furthermore we maintain that Grid applications can be built with today's Web Services technology.

References

[1] "Axis." http://ws.apache.org/axis/.
[2] "Service-Oriented Architecture (SOA) Definition." http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.
[3] "SETI@home." http://setiathome.ssl.berkeley.edu/.
[4] "Web Services Addressing (WS-Addressing)." http://msdn.microsoft.com/ws/2004/03/ws-addressing.
[5] "Web Services Policy (WS-Policy)." http://msdn.microsoft.com/ws/2002/12/policy.
[6] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, A. Nadalin, N. Nagaratnam, M. Nottingham, H. Prafullchandra, C. v. Riegen, J. Schlimmer, C. Sharp, and J. Shewchuk, "Web Services Policy Framework (WS-Policy)." http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-policy.asp, 2004.
[7] Berkeley, "SETI@Home." http://setiathome.ssl.berkeley.edu/.
[8] P. Emeagwali, "Metaphysics of the Grid," vol. 2. http://www.gridtoday.com/03/0721/101710.html, 2003.
[9] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, pp. 115-150, 2002.
[10] I. Foster and D. Gannon, "Open Grid Services Architecture Platform (OGSA)," https://forge.gridforum.org/projects/ogsa-wg, 2003.
[11] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Global Grid Forum 22 June 2002.
[12] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organisations," *International Journal of Supercomputer Applications*, vol. 15, 2001.
[13] H. He, "What is Service-Oriented Architecture." http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html, 2003.
[14] HP, "Grid & Utility Computing." http://devresource.hp.com/drc/topics/utility_comp.jsp.
[15] IBM, "On demand computing." http://www-1.ibm.com/grid/.
[16] Microsoft, "Web Services Enhancements (WSE)." http://msdn.microsoft.com/webservices/building/wse.
[17] Microsoft, "Web Services Specifications Index." http://msdn.microsoft.com/webservices/understanding/specs/.
[18] Microsoft, "Application Conceptual View." http://msdn.microsoft.com/library/en-us/dnea/html/eaappconland.asp, 2002.
[19] Microsoft, "Indigo." http://msdn.microsoft.com/Longhorn/understanding/pillars/Indigo, 2004.
[20] Microsoft, "Remarks by Bill Gates, Chairman and Chief Software Architect, Microsoft Corporation" Seamless Computing: Hardware Advances for a New Generation of Software" Windows Hardware Engineering Conference (WinHEC) 2004." http://www.microsoft.com/billgates/speeches/2004/05-04winhec.asp, 2004.
[21] M. Mullender and M. Burner, "Application Conceptual View." http://msdn.microsoft.com/architecture/application/default.aspx?pull=/library/en-us/dnea/html/eaappconland.asp: Microsoft, 2002.
[22] OASIS, "Web Services Composite Application Framework (WS-CAF)." http://www.oasis-open.org/committees/ws-caf.

[23] ORACLE, "Oracle Grid Computing." http://www.oracle.com/technologies/grid/.
[24] S. Parastatidis, J. Webber, and P. Watson, "Using Web Services to Build Grid Applications - The "No Risk" WSGAF Profile." http://www.neresc.ac.uk/ws-gaf/WSGAF-NoRiskProfile.pdf, 2004.
[25] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "A Grid Application Framework based on Web Services Specifications and Practices." http://www.neresc.ac.uk/ws-gaf, 2003.
[26] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "WS-GAF: A Grid Application Framework based on Web Services Specifications and Practices." Submitted for publication, 2004.
[27] S. Parastatidis, J. Webber, S. Woodman, D. Kuo, and P. Greenfield, "SOAP Service Description Language (SSDL)," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-899, 2005.
[28] D. Sprott and L. Wilkes, "Understanding Service-Oriented Architecture." http://msdn.microsoft.com/library/en-us/dnmaj/html/aj1soa.asp, 2004.
[29] W. Vogels, "Web Services Are Not Distributed Objects," *IEEE Internet Computing*, vol. 7, pp. 59-66, 2003.
[30] W3C, "Semantic Web." http://www.w3.org/2001/sw/.
[31] W3C, "SOAP Version 1.2 Part 1: Messaging Framework." http://www.w3.org/TR/soap12-part1.
[32] W3C, "Web Services Description Language (WSDL)." http://www.w3.org/2002/ws/desc.
[33] W3C, "Web Services Architecture." http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 2004.
[34] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "A Note on Distributed Computing," Sun Microsystems, Mountain View, CA SMLI TR-94-29, 1994.
[35] J. Webber and S. Parastatidis, "The WS-GAF Registry Service," presented at Building Service-based Grids Workshop (GGF 11), Honolulu, Hawaii, 2004.
[36] WS-I, "Web Services Interoperability (WS-I) Interoperability Profile 1.0a." http://www.ws-i.org.