How Do I Model State? Let Me Count

A study of the technology and sociology of Web services specifications

There is nothing like a disagreement concerning an arcane technical matter to bring out the best (and worst) in software architects and developers. As every reader knows from experience, it can be hard to get to the bottom of what exactly is being debated. One reason for this lack of clarity is often that different people care about different aspects of the problem. In the absence of agreement concerning the problem, it can be difficult to reach an agreement about the solutions.

In this article we discuss a technical matter that has spurred vigorous debate in recent years: How to define interactions among Web services to support operations on state (that is, data values associated with a service that persist across interactions, so that the result of one operation can depend on prior ones).[4] An airline reservation system and a scheduler of computational jobs are two examples of systems with this requirement. Both must provide their clients with access to information about ongoing activities: reservations and jobs, respectively. Clients typically want to name and/or identify state (refer to a specific reservation or job), access that state (get the status of a flight reservation or the execution progress of a job), modify part of that state (change the departure time of a flight or set the CPU requirements of a job), and destroy it (cancel a reservation or kill a job).

Ian Foster, Argonne National Laboratory
Savas Parastatidis, Microsoft Research
Paul Watson, Newcastle University
Mark McKeown, University of Manchester

the Ways

How Do I Model State? Let Me Count the Ways

The debate over this issue does not concern the need for such operations but rather the specifics of how exactly to model and implement service state and the associated interactions on that state. State may be modeled *explicitly* by the distributed computing technology used (for example, as an "object" with create, read, update, and destroy operations) or *implicitly* by referring to application domain-specific concepts within the interactions (for example, "create reservation" and "update reservation" messages that include a domain-specific identifier such as an ASIN—Amazon standard identification number—in the body). Along a different dimension, we may use HTTP or SOAP as an implementation technology.

Our goal here is to shed light on possible approaches to modeling state. To this end, we present four different approaches and show how each can be used to enable access to a simple job management system. Then we summarize the key arguments that have been made for and against each approach. In addition to providing insights into the advantages and disadvantages of the different approaches, the discussion may also be interesting as a case study in technical debate. As we will see, the four approaches are remarkably similar in what they do, but differ in precisely how they do it.

Alphabet and Specification Soup

Any discussion of Web services inevitably involves a plethora of acronyms and specification names. We list some of them here. To save space, we do not provide citations for individual specifications. These can easily be located online.

EPR (endpoint reference). As defined in the WS-Addressing specification, a combination of Web services elements that define the address for a resource in a SOAP header.

SOAP. A protocol for exchanging XML-based messages over networks, normally using HTTP.

WSDL (Web Services Description Language). An XML-based language that provides a model for describing Web services.

WS-Eventing. A specification that defines a protocol for Web services to subscribe to another Web service or accept a subscription from another Web service.

WSDM (Web Services Distributed Management) An OASIS-developed Web services architecture and set of specifications for managing distributed resources.

WS-ResourceTransfer. A proposed integration of WS-RF and WS-Transfer.

WS-RF (Web Services Resource Framework). An OASIS-developed architecture and set of specifications for describing and accessing state in a Web service.

WS-Transfer. A specification that defines a protocol for the transfer of an XML representation of a WS-addressable resource, as well as for creating and deleting such resources.

SOME PRELIMINARY OBSERVATIONS

First, a few observations about what we mean by *modeling state*. The systems with which we want to interact may have simple or complex internal state. Various aspects of this state may be exposed so that external clients can engage in "management" operations. For example, an airline reservation system might give customers the ability to programmatically create, monitor, and manage reservations. The same system might also allow operators to programmatically access information about current system load and the mapping of computational resources to different system functions. We are not suggesting these mechanisms provide direct access to the underlying state in its entirety. Rather, we are assuming the principles of encapsulation and data integrity/ownership are maintained. It is up to a system's designer to define the projections to those aspects of the system's internal state that they are willing to expose to the outside world.[5]

Such projections can be complex. In the case of a job management system, for example, the underlying state associated with even an apparently simple job may consist of multiple distinct processes on different back-end computers, entries in various internal tables and catalogs, and activities within subsystems such as schedulers and monitors. When designing the allowed interactions with such a system, we must model the "state of a job" (the projection of the complex underlying state that is to be made available to clients) in a manner that is not only easy for clients to understand and use but that also makes it possible to maintain this projection effectively.

We use the shorthand "modeling state" rather than the unwieldy "modeling a projection of underlying system state." It is important to bear in mind, however, the reality of what could be going on behind the boundaries of a system with which an interaction takes place.

We also make a few remarks concerning the difference between architectural styles and implementation technologies. The evolution of the Web from an infrastructure that enables access to resources to a platform for distributed applications has resulted in much discussion on the relevant architectural approaches and technologies. Terms such as *REST* (representational state transfer,[3] an architectural style) and *HTTP* (a protocol specification) are often used interchangeably to indicate an architectural approach in which a small set of verbs/operations (PUT, GET, DELETE) with uniform semantics are used to build applications. Similarly, the popularity of *Web services* (a set of protocol specifications)[1] has resulted in the use of that term as a synonym for service orientation (an architectural style).

We draw a distinction between the architectural styles and their implementation technologies. Instances of the former represent a collection of principles and constraints that provide guidance when designing and implementing distributed applications. In contrast, the latter are the mechanisms or tools used to apply the principles of an architectural style when building applications. There is not a one-to-one mapping between an architectural style and an implementation technology, even though one set of tools may be easier to use when applying a particular set of principles. For example, pure HTTP is particularly well suited for implementing distributed applications according to REST principles, while Web services technologies such as SOAP are better suited for interface-driven applications. There is no reason, however, why one could not build a REST-oriented application using Web services technologies or a distributed object-based application using HTTP—although we doubt anyone would want to go through such an exercise.

FOUR APPROACHES TO MODELING STATE

Table 1 summarizes the key properties of the four approaches presented here. The following provides a brief description of each approach.

WS-RF APPROACH

WS-RF (Web Services Resource Framework) defines conventions on how state is modeled and managed using Web services technologies. WS-RF implements an architectural style similar to that of distributed objects or resources. Projected state is explicitly modeled as an XML document (the state representation) and is addressable via a WS-Addressing EPR (endpoint reference), a conventional representation of the information that a client needs to access a network service.

As in traditional object-based systems, any number of operations can be defined that access, or result in the change of, the projected state. The WS-RF specifications, however, define a set of common operations for the following: accessing that projected state (the XML document) in its entirety or in part; requesting notification of changes on it (using WS-Notification); updating it in its entirety or in part; and deleting it. The structure of the XML document (that is, the schema), together with all the operations that can be applied to the projected state, known as the resource, are included in the WSDL (Web Services Description Language) document associated with the state's EPR, thus allowing clients to discover, using standard operations, what state a particular service makes available.

The WS-RF and WS-Notification specifications were developed within OASIS (Organization for the Advancement of Structured Information Standards). They are implemented within various open source and proprietary systems. Other specifications, notably WS-Notification and WSDM (Web Services Distributed Management), build on WS-RF.

WS-TRANSFER APPROACH

WS-Transfer, like WS-RF, models the projected state explicitly through an XML document accessible via an EPR. However, the only operations defined on that state are, as per the CRUD (create, retrieve, update, and delete) architectural style: *create* a new resource state representation by supplying a new XML document; *get* an entire resource state representation; *put* a new resource state representation to replace an existing one; and *delete* an existing state representation. Distributed, resource-oriented applications are then built by using these operations to exchange state representations.

The WS-Transfer specification was developed by an industry group led by Microsoft and has recently been submitted to the W3C (World Wide Web Consortium) for standardization. Other specifications, notably WS-Eventing and WS-Management, build on WS-Transfer. As we will see later, WS-Transfer and WS-RF differ only in minor technical details; they arguably owe their separate existence more to industry politics than technical considerations. Fortunately, there seems to be industry support for an integration of the WS-Transfer and WS-RF approaches, based on a WS-Transfer substrate—the WS-ResourceTransfer specifications.

HTTP APPROACH

HTTP is an application protocol implementing a resource-oriented approach to building distributed systems. It has been described as an implementation of the REST architectural style. Like WS-RF and WS-Transfer, HTTP implements a resource-oriented approach to building distributed systems. According to REST, a small set of verbs/operations with uniform semantics should be used to build hypermedia applications, with the Web being an example of such an application. The constraints applied

TABLE 1 **Key Characteristics of the Four Approaches**

	WF-RF	WS-Transfer	HTTP	No conventions
State representation schema	WSDL extensions			
Address state representation	EPR (WS-Addressing)	EPR (WS-Addressing)	URI	URN
Create new state		Create (WS-Transfer)	HTTP POST	
Access entire state	GetResourcePropertyDocument (WS-ResourceProperties)	Get (WS-Transfer)	HTTP GET	
Get part of state	GetResourceProperty, GetMultipleResourceProperties, QueryResourceProperties (WS-ResourceProperties)		Not defined unless part of a state representation is exposed through a different URI (no semantics about the relationship are defined)	
Update entire state	SetResourceProperties (WS-ResourceProperties)	Put (WS-Transfer)	HTTP PUT	
Update, or add, part of state	SetResourceProperties, InsertResourceProperties, UpdateResourceProperties, DeleteResource-Properties (WS-ResourceProperties)			
Request notification	Subscribe (WS-Notification)	Subscribe (WS-Eventing)		Subscribe (WS-Eventing)
Lease-based life-time management	SetTerminationTime (WS-resourcelifetime)			
Destroy state	Destroy (WS-ResourceLifetime)	Delete (WS-Transfer)	HTTP DELETE	
Fault modeling	Well-defined error codes (WS-BaseFaults + other specs)	SOAP faults	HTTP fault codes	SOAP faults
RPC-based	Yes	Yes	No	No
Open standards process	Yes (OASIS)	Yes (W3C)	Already a standard	No need for new standards

through the semantics of these operations aim to allow applications to scale (for example, through caching of state representations). State representations are identified through a URI.

HTTP defines simple verbs—such as POST, PUT, GET, DELETE—and headers to enable the implementation of applications according to REST principles. XML is just one of the many media formats that HTTP can handle.

NO-CONVENTIONS APPROACH

Finally, in the "no conventions for managing state" approach (shortened to "no conventions" in table 1),[6] there are no such concepts as operations, interfaces, classes, state, clients, or servers. Instead, applications are built through the exchange of one-way messages between services. Semantics to the message exchanges (for example, whether a message can be cached or whether a transactional context is included) are added through composable protocols. State representations are not fundamental building blocks. Instead, resources should be identified through URIs (or URNs) inside the messages, leaving it up to the application domain-specific protocols to deal with state management. Although any asynchronous messaging technologies could be used in implementations following this style, we consider here an implementation based on Web services protocols, but without the introduction of state-related conventions.

JOB MANAGEMENT EXAMPLE

We use a simple example to provide a more concrete comparison of these four approaches. The example is a job management system that allows clients both to request the creation of computational tasks ("jobs") and to monitor and control previously created jobs. It provides the eight operations listed in table 2, which we choose to represent as a range of typical state manipulation operations. In each case, a client makes the request by sending an appropriate message to the job management system and then expects a response indicating success or failure.

Operation 1 creates a new job and returns a *handle* that can be used to refer to the job in subsequent operations. Parameters specify such things as required resources, an initial lifetime for the job, and the program to be executed.

Operations 2–7 support some archetypal job monitoring and control functions on a single job.

Operation 8 is an example of an interaction that may relate to multiple jobs. The set of jobs to which the operation is to be applied might be specified either in terms of job characteristics or by supplying a set of job handles.

In the discussion that follows, we show how our four approaches can be used to build a service that supports these eight tasks. Each approach not only has in common that the "job factory service" is a network endpoint to which job creation and certain other requests should be directed, but also is distinguished from the other approaches in terms of:

• Its *syntax* (that is, how the job handle should be represented and how operations on the job should be expressed in messages).

• Its use (or not) of *conventions* defined in existing specifications for the purpose of defining its syntax.

The distinction made here between syntax and conventions may appear unimportant, but we emphasize it so that we can focus in this section on syntax and postpone discussion of the advantages or disadvantages of

TABLE 2 **The Eight Operations Considered in Our Comparison of Approaches**

#	Operation	Description
1	Create new job	A client requests the creation of a new job by sending a job creation request to a job factory service responsible for creating new jobs. Upon success, a job handle is returned that can be used to refer to the job in subsequent operations.
2	Retrieve state	Retrieve all state information associated with a specified job (e.g., execution status, resource allocation, program name).
3	Status	Determine the execution status (e.g., active, suspended) of a specified job.
4	Lifetime	Extend the lifetime of a specified job.
5	Subscribe	Request notification of changes in the state of a specified job.
6	Suspend	Suspend a specified job.
7	Terminate	Terminate a specified job.
8	Terminate multiple	Apply the terminate operation to all jobs that satisfy certain criteria, such as those belonging to a particular client, those with a particular total execution time, those with a specific current execution status, or those with explicitly identified job handles.

How Do I Model State? Let Me Count the Ways

TABLE 3 **Syntax used in WS-RF Job Management Interface**

#	Message	To	Returns on success
1	CreateJob(job-specification)	job-factory-service	WS-Resource-qualified EPR to job state (job handle)
2	**GetResourcePropertyDocument**()	job-handle EPR	XML document comprising all job state
3	**GetResourceProperty**("status")	job-handle EPR	Job status
4	**SetTerminationTime**(lifetime)	job-handle EPR	New lifetime
5	**Subscribe**(condition)	job-handle EPR	EPR to subscription
6	Suspend	job-handle EPR	Acknowledgment
7	**Destroy**	job-handle EPR	Acknowledgment
8	DestroyMultiple(job-description)	job-factory-service	Acknowledgment

TABLE 4 **Syntax used in WS-Transfer Job Management Interface**

#	Message	To	Returns on success
1	**Create**(job-spec)	job-factory-service	EPR to job state
2	**Get**()	job-handle EPR	XML document comprising all job state
3	**Get**()	job-handle EPR	XML document comprising all job state, from which status can be extracted.
4	Setlifetime(lifetime)	job-handle EPR	New lifetime
5	**Subscribe**(condition)	job-handle EPR	EPR to subscription
6	Suspend	job-handle EPR	Acknowledgment
7	**Delete**()	job-handle EPR	Acknowledgment
8	DeleteMultiple(job-spec)	job-factory-service	Acknowledgment

TABLE 5 **Syntax used in REST Job Management Interface**

#	Message	To	Returns on success
1	**HTTP POST** *job-specification*	job-factory-service (e.g., http://grid.org)	URI(s) identifying the job(s) that have been created (e.g., http://grid.org/bloggs/Jobs/4523)
2	**HTTP GET**	http://grid.org/bloggs/Jobs/4523	A representation of the state of the job, including Xlinks and semantic information
3	**HTTP GET**	http://grid.org/bloggs/Jobs/4523/status	A representation of the status of the job
4	**HTTP PUT** *exp-time*	http://grid.org/bloggs/Jobs/4523/lifetime	Acknowledgment
5	**HTTP POST** *condition*	http://grid.org/bloggs/Jobs/4523/subs	URI identifying the new subscription
6	**HTTP PUT** "suspended"	http://grid.org/bloggs/Jobs/4523/status	Acknowledgment
7	**HTTP DELETE**	http://grid.org/bloggs/Jobs/4523	Acknowledgment
8	—	—	—

adopting specific "standards" (conventions) to later in this article.

WS-RF implementation. Table 3 describes a job management interface based on the WS-RF and WS-Notification specifications. We use boldface type to indicate operation names that are defined in some specification associated with the approach in question. Those operations that are not in boldface are, by definition, not defined in any existing specification, and thus their syntax and semantics represent somewhat arbitrary choices, selected for illustrative purposes. We see that WS-RF and WS-Notification specifications provide five of the eight required functions.

The job handle returned upon success from operation 1 is represented as an EPR. A client receiving such a job handle can then use it as a destination for operations 2–7. Note that requests are directed to the Web services address contained in the job handle EPR, which may or may not be the job factory service. This distinction is important because it allows for a logical and/or physical separation between the job factory and job management functions.

Operation 8 is sent directly to the job factory service, which is assumed to have access to information about all active jobs. The argument could be, for example, a specification (such as an XPath specification) identifying the jobs that are to be terminated (for example, all jobs created by Bloggs or all jobs that have exceeded their quota, and/or a list of EPRs denoting the jobs to be terminated).

WS-Transfer implementation. Table 4 describes a job management interface based on the use of the WS-Transfer and WS-Eventing specifications, which provide five of the eight required operations. As in the WS-RF interface, the job handle returned upon success from operation 1 is represented as an EPR, and a client receiving such a job handle can then use it as a destination for operations 2–7. Note that requests are directed to the Web services address contained in the job handle structure, which may or may not be the job factory service.

An alternative treatment of operation 3 is possible, albeit with some extra work, avoiding the need to transfer the entire state document. A new operation (for example, GetEPRtoPart) is defined, requesting that a new state representation be exposed, through a different EPR, representing parts of the original state representation. The Get() operation is then applied to this new EPR. WS-Transfer applications (and higher-level specifications such as WS-Management) often use this approach to address the lack of support for partial state access in WS-Transfer.

Operation 8 is sent directly to the job factory service, which is assumed to have access to information about all active jobs, as in the WS-RF case.

HTTP implementation. Table 5 summarizes the syntax of an HTTP implementation. Note that operation 5 can alternatively be addressed via some custom encoding or by using a system such as SMTP, Jabber, SMS, or Atom. HTTP DELETE cannot take any content, so there is no way to specify that a set of jobs (operation 8) can be deleted by using the HTTP DELETE message, except in the case when we delete all jobs in some predefined group (for example, "HTTP DELETE http://grid.org/Bloggs/Jobs" to delete all jobs created by Bloggs).

Note that whereas HTTP defines all the verbs used, the

TABLE 6 **Syntax Used in Job Handle Approach**

#	Message	To	Returns on success
1	"New job" carrying the job specification	job-management-service	Identifier for the newly created job
2	"State" carrying job identifier(s)	any job service aware of the job identifier(s)	XML document with job state(s)
3	"Status" carrying job identifier(s)	any job service aware of the job identifier(s)	XML document with job status(s)
4	"New lifetime" carrying job identifier(s) and new lifetime(s)	any job service aware of the job identifier(s)	Acknowledgment*
5	"Subscription" carrying job identifier(s) and subscription information (for each)	any job service aware of the job identifier(s)	Acknowledgment*
6	"Suspension" carrying job identifier(s)	any job service aware of the job identifier(s)	Acknowledgment*
7	"Destroy request" carrying job identifiers(s)	any job service aware of the job identifier(s)	Acknowledgment*
8	"Destroy request" carrying job identifier(s) and query expression	any job service aware of the job identifier(s)	Acknowledgment*

* *May also silently accept the message and report on fault only. If proof of delivery is necessary, WS-ReliableMessaging can be used.*

structure of the URIs and the format and semantics of the documents exchanged in order to implement the job service's operations are application specific. Thus, while the URIs appear to convey some semantic information based on their structure (for example, a /status at the end of a particular job URI may be interpreted by a human as the identifier of the status resource), this is an application-specific convention.

No-conventions/Web services implementation. Since in this approach we assume no defined state management conventions with widely agreed-upon semantics, each application domain is expected to define interactions that meet its own requirements. Table 6 summarizes a potential implementation of the job management service in this style, using SOAP messaging.

Because of the nature of the example chosen, all operations are defined as request-response message exchanges. The CreateJob message exchange returns an identifier as the job handle. The returned job identifier may be a globally unique URI (for example, a URN) that can be accepted by multiple job services. Metadata about it (for example, the job services that "know" about it) may be discovered from registries. Examples of this approach to identifying resources include the LSID (Life Science Identifier), IVOA (International Virtual Observatory Alliance) identifier, and ASIN. Thus, the job identifier may become a technology-independent handle that can also be used with other technologies (for example, a Jini or CORBA interface to the same service). A client receiving such a job handle can then pass it to the job management service.

DISCUSSION

The four approaches to modeling state do not differ greatly in terms of what they actually do. All send essentially the same messages, with the same content, across the network. For example, a request to destroy a particular job will in each case be directed to a network endpoint via an HTTP PUT, and will contain the name of the operation to be performed, plus some data indicating the job that should be destroyed. The approaches vary only in how these different components are included in a message, an issue that may have implications for how messages can be processed and routed but that has no impact on how services are implemented.

At the same time, there clearly are significant differences among the approaches in their use of conventions, the underlying protocols, and the tooling available to support their use. Here we summarize important arguments on these topics. The characterizations of the various positions are our own.

The value of convention. Proponents of the WS-RF and WS-Transfer approaches argue that creating, accessing, and managing state involve a set of common patterns that can usefully be captured in a set of specifications, thus simplifying the design, development, and maintenance of applications that use those patterns. For example, in the case of our job management service, these proponents might observe the following:

• The creation and subsequent management of a job can naturally be viewed as an instance of some general patterns (creation, access, subscription, lifetime management, and destruction of state associated with a job).

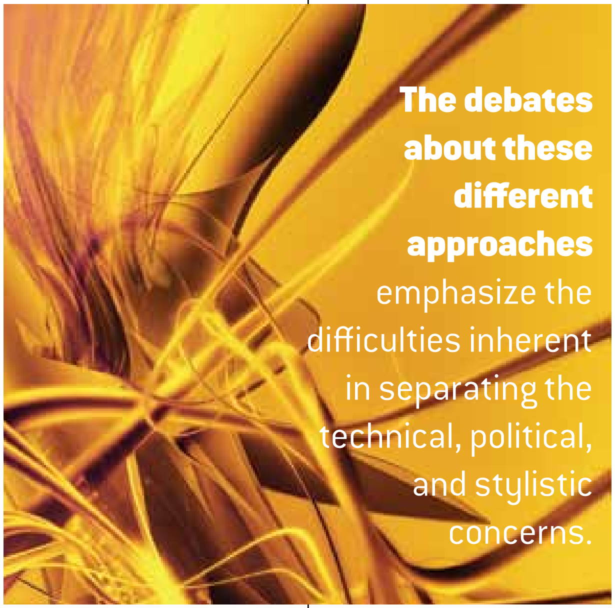
• The encoding of the job management interface in terms of those patterns simplifies both the design of the interface (as much of it is already provided in other specifications) and the explanation of that interface to others.

They may also point out that programmer productivity is enhanced by client tools and applications that are "WS-RF and/or WS-Transfer aware." For example, a registry or monitoring system can use WS-RF operations to access service state without any application-specific knowledge of that state's structure.[2]

In contrast, proponents of the no-conventions approach argue that the design of any particular interface (for example, one for job management) will typically be relatively simple, involve issues that are not captured by these conventional patterns, and not require all features included in the specifications that encode that pattern. Thus, one can achieve a simpler design by proceeding from first principles. For example, in the case of our job management service, while WS-RF and WS-Transfer provide us with a Destroy operation "for free," we still need to introduce a separate Suspend operation. Furthermore, the semantics of Destroy may be quite application specific. For example, a service implementer may decide to retain information about destroyed jobs (by changing their status to "destroyed") so that information about them can still be retrieved. In this case, the state would not be destroyed.

Finally, WS-RF and WS-Transfer focus on interaction with single states (for example, DestroyMultiple had to be custom defined) and so may not provide useful conventions in cases where operating on multiple states is the common case; for example, all Amazon REST and Web services can consume multiple ASINs.

Ideally, we would like to evaluate the relative merits of these two positions in terms of concrete metrics such as code size. Such an evaluation, however, requires agreement on the requirements that the interfaces should support. Unfortunately, proponents of the different approaches tend to differ also in their views of requirements. For example, a proponent of common patterns might see the ability to use WS-Resource Lifetime operations for soft-state lifetime management as desirable, while others might not see that feature as important.



The debates about these different approaches emphasize the difficulties inherent in separating the technical, political, and stylistic concerns.

Standards. Another topic of disagreement concerns the importance of standardized specifications. Unfortunately (but not uncommonly in the world of Web services), we are faced with not one but two sets of Web services specifications, similar in purpose and design but different in minor aspects of syntax and semantics. WS-RF has been submitted to, reviewed, and approved by a standards body (OASIS); WS-Transfer has been submitted to W3C but is not yet a W3C recommendation. The proposed consolidation of the two, WS-ResourceTransfer, adds to the confusion. Thus, we get a mix of opinions, including the following:

• WS-RF, having undergone intensive community review by a standards body, is therefore technically superior and/or morally preferable to the "proprietary" WS-Transfer.
• WS-RF is superior to WS-Transfer for technical reasons—for example, its support for access to subsets of a resource's state, which can be important if that state is large. (In WS-Transfer, the same effect can be achieved, but only by defining an auxiliary operation that returns an EPR to a desired subset.)
• WS-Transfer is superior to WS-RF for technical reasons—for example, its greater simplicity.
• As a general principle, we should employ only specifications that are stable, widely accepted, and supported by interoperable tooling. Because neither WS-RF nor WS-Transfer is supported by all major vendors, neither passes this test. This view argues for the no-conventions approach.
• HTTP is widely used on the Web and, as a result, it should be preferred over any WS-based solution.

Implementation reuse. Proponents of conventions such as WS-RF and WS-Transfer argue that the adoption of conventional patterns can facilitate code reuse. Every WS-RF or WS-Transfer service performs such things as state access, lifetime management, concurrency control for incoming requests, and state activation/deactivation in the same way. Thus, the code that implements those behaviors can be reused in service implementations.

Proponents of the no-conventions approach, however, reply that service implementers can also use the same code. In other words, there is certainly value to providing service implementers with standardized implementations of common tasks, but this need not imply that those patterns are exposed outside the service.

Simplicity vs. structure. Proponents of the HTTP/REST approach emphasize that it provides for more concise requests and permits the use of simpler client tooling than approaches based on Web services. Critics point out that the use of HTTP/REST means that users cannot leverage the significant investment in Web services technologies and platforms for message-based interactions.

SUMMARY

We have presented four different approaches to modeling state in Web services interactions. Each approach defines roughly comparable constructs for referring to, accessing, and managing state components, but differs according to both its precise syntax and the use made (or not) of conventional domain-independent encodings of operations.

Thus, when defining state management operations, the WS-RF and WS-Transfer approaches both use EPRs to refer to state components and to adopt conventions defined in the WS-RF and related specifications and in the WS-Transfer and related specifications, respectively. In contrast, the no-conventions and REST approaches adopt domain-specific encodings of operations, on top of SOAP and HTTP, respectively.

Analysis of the debates that have occurred around these different approaches emphasizes the difficulties inherent in separating technical, political, and stylistic concerns. Some differences of opinion relate to well-defined technical issues and reflect either different philosophies concerning system design or different target applications. Others relate to differing target time scales. For example, no-conventions proponents initially argued against the use of WS-Addressing because of lack of support for that specification in certain tools, while WS-RF and WS-Transfer proponents argued in favor, believing that WS-Addressing would eventually become universal. Support for WS-Addressing has since become quasi-universal, and now few find its use objectionable. Q

REFERENCES

1. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. 2003. *Web Services Architecture*. W3C.
2. Chervenak, A., Schopf, J.M., Pearlman, L., Su, M.-H., Bharathi, S., Cinquini, L., D'Arcy, M., Miller, N., Bernholdt, D. 2006. Monitoring the Earth System Grid with MDS4. IEEE Conference on e-Science and Grid Computing. Amsterdam, Netherlands.
3. Fielding, R. 2000. Architectural styles and the design of network-based software architectures. *Information and Computer Science*. University of California Irvine.
4. Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D., Tuecke, S. 2005. Modeling and managing state in distributed systems: The role of OGSI and WSRF. *Proceedings of the IEEE* 93(3): 604–612.
5. Helland, P. 2004. *Data on the Inside vs. Data on the Outside*. Microsoft.
6. Parastatidis, S., Webber, J., Watson, P. and Rischbeck, T. 2005. WS-GAF: A framework for building grid applications using Web services. *Concurrency and Computation: Practice and Experience* 17 (2–4): 391–417.

ACKNOWLEDGMENTS

We are grateful to Karl Czajkowski, Jim Gray, and Sam Meder for comments on an earlier version of this document. Needless to say, the characterizations of the different arguments are our own. The work of the first author was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. The work at Newcastle was supported by the UK eScience Programme, with funding from the EPSRC, DTI, and JISC.

LOVE IT, HATE IT? LET US KNOW
feedback@queue.acm.org

IAN FOSTER (foster@anl.gov) is the director of the Computation Institute at Argonne National Laboratory, where he is an Argonne Distinguished Fellow, and the University of Chicago, where he is the Arthur Holly Compton Distinguished Service Professor of computer science.
SAVAS PARASTATIDIS (Savas.Parastatidis@newcastle.ac.uk) is an architect for Microsoft Research. He investigates the use of technology in e-research and is particularly interested in cloud computing, knowledge representation and management, and social networking.
PAUL WATSON (Paul.Watson@newcastle.ac.uk) is a professor of computer science at Newcastle University, and director of the North East Regional e-Science Centre in the U.K.
MARK McKEOWN was a grid architect at the University of Manchester, U.K., at the time of this work.

This article appears in print in the September 2008 issue of *Communications of the ACM*.

ACM Digital Library

www.acm.org/dl

The Ultimate Online INFORMATION TECHNOLOGY Resource!

Powerful and vast in scope, the **ACM Digital Library** is the ultimate online resource offering unlimited access and value!

The **ACM Digital Library** interface includes:

- **The ACM Digital Library** offers over 40 publications including all ACM journals, magazines, and conference proceedings, plus vast archives, representing over 2 million pages of text. The ACM DL includes full-text articles from all ACM publications dating back to the 1950s, as well as third-party content with selected archives. PLUS NEW: Author Profile Pages with citation and usage counts and New Guided Navigation search functionality! **www.acm.org/dl**

- **The Guide to Computing Literature** offers an enormous bank of over one million bibliographic citations extending far beyond ACM's proprietary literature, covering all types of works in computing such as journals, proceedings, books, technical reports, and theses! **www.acm.org/guide**

- **The Online Computing Reviews Service** includes reviews by computing experts, providing timely commentary and critiques of the most essential books and articles.

Available only to ACM Members.
Join ACM online at **www.acm.org/joinacm**

To join ACM and/or subscribe to the Digital Library, contact ACM:

Phone:	1.800.342.6626 (U.S. and Canada)
	+1.212.626.0500 (Global)
Fax:	+1.212.944.1318
Hours:	8:30 a.m.-4:30 p.m., Eastern Time
Email:	acmhelp@acm.org
Join URL:	www.acm.org/joinacm
Mail:	ACM Member Services
	General Post Office
	PO Box 30777
	New York, NY 10087-0777 USA

*Guide access is included with Professional, Student and SIG membership. ACM Professional Members can add the full ACM Digital Library for only $99 (USD). Student Portal Package membership includes the Digital Library. Institutional, Corporate, and Consortia Packages are also available.

acm Association for Computing Machinery

Advancing Computing as a Science & Profession

AD29