Turning Software into a Service

The software as a service model composes services dynamically, as needed, by binding several lower-level services—thus overcoming many limitations that constrain traditional software use, deployment, and evolution.

Mark Turner

David Budgen

Pearl Brereton

Keele University, Staffordshire

I n 1999, the Pennine Group—a consortium of software engineering researchers from the University of Durham, Keele University, and the University of Manchester Institute of Science and Technology—asserted that continued development of new architectural styles based on constructional forms such as objects or components would not advance the software field. Rather, they foresaw a future in which developers take a radically different view of how software delivers its functionality to users.[1]

We have explored the *software as a service* (SaaS) concept through several small-scale experiments.[2] This concept envisages a demand-led software market in which businesses assemble and provide services when needed to address a particular requirement. The SaaS vision is a vital contribution to current thinking about software development and delivery that has arisen in part from initiatives in the Web services and electronic-business-communication communities.

SaaS focuses on separating the *possession* and *ownership* of software from its *use*. Delivering software's functionality as a set of distributed services that can be configured and bound at delivery time can overcome many current limitations constraining software use, deployment, and evolution. Such a model would open up new markets, both for relatively small-scale specialist-services providers and for larger organizations that provide more general services. In addition, service provision could include the dynamic creation and development of entirely new services that use existing ones. The "Sample SaaS Scenario" sidebar shows inherent SaaS ideas in the context of a company helping with an overseas property purchase.

This approach lets the set of services a business uses evolve without any user intervention as that business and its context change. Also, the key know-how involved is not *who* provides services, necessary though that knowledge is, but *what* service a transaction requires at any particular point, along with negotiating suitable terms for its use. Selecting and binding the means of providing an appropriate service can therefore be performed dynamically, on demand, through *ultra-late binding*.

SAAS AND OTHER SERVICE FORMS

Despite advances in programming language and development environment technologies, the basic paradigm for constructing and maintaining software has altered little since the 1960s. Developers still construct software largely by employing some variant of the *edit-compile-link* cycle to generate an executable binary image from a source described using a procedural programming language. Although the Web may have widened our interpretation of what software is, the practices used to develop and implement a Web site differ little from those traditionally employed for constructing software and are just as error prone.

To achieve our vision, we have focused on developing a radically different paradigm. We believe that software should deliver a *service*. Further,

Sample SaaS Scenario

The following scenario, set within the context of a much larger example, demonstrates the ideas inherent in the software-as-a-service concept.

Alice has set up a company that offers services to people who want to purchase property abroad. She currently offers two services: One provides information about available properties, while a second handles actual negotiation and purchase.

Alice's purchasing service uses other services to handle tasks such as translation, legal and financial negotiations, financing, and currency transfer. Offering these services involves specifying the terms, conditions, and form of service provision, together with the rules describing how other services will be employed. This is the know-how element of her service.

Drawn from the complete scenario, one small task concerning a legal document might play out as follows. Alice's purchasing service urgently needs to have the document translated from Spanish to English, which involves the following steps:

- The purchasing service seeks a translation service and, after negotiating the terms and conditions, selects Scribe, the cheapest from the four available.
- Scribe, a broker that evaluates services but doesn't actually provide them, seeks a Spanish translation service geared to handling legal documents. From the three providers offering this service, based upon previous use history and satisfactory delivery, Scribe selects ES-trans, which offers an immediate service.
- ES-trans provides the translation that Alice's purchasing service requires.

Within SaaS, the services involved in a system can change with the associated business, and SaaS can perform this change dynamically. For example, in Alice's scenario, such a change might occur if Scribe becomes aware of new translation services.

A more complex example, involving composition, would be for Alice to add a new service to help negotiate mortgages. Relating this example to the key service-oriented functions of our service integration layer can be instructive.

First, in terms of service description, Alice must be able to describe her general requirement to have a document translated, its more specific details—including that it is a *legal* document—and the languages involved. Service description also encompasses the way a service provider such as Scribe or ES-trans describes the services it can supply—either directly or using other providers—and the parameters within which they will negotiate a service contract.
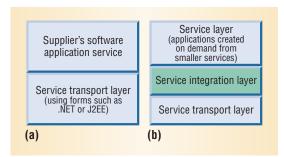
This example contains two instances of service discovery. The first occurs when Alice seeks the translation service, the second when the Scribe broker seeks a service that can deliver the translation in the required time.

Likewise, the example shows two stages of negotiation. First, Alice's service will negotiate with Scribe. Because she frequently needs translation services in general, however, the service also may conduct this negotiation periodically, resulting in a longer-term contract between Alice's service and Scribe, in which case the immediate negotiation will be concerned purely with service parameters. Second, Scribe can then negotiate a much shorter-term, per-document agreement with ES-trans and other possible providers.

Service delivery in this example occurs when ES-trans receives the original document and returns the translation. However, if ES-trans does not do this within the specified time—determined by the form in which Alice's service may have defined "urgently" in this case—either Alice's service or Scribe can opt to invoke the suspension step and renegotiate with another provider. Finally, a form of service composition occurs when Scribe employs its know-how about translation services to seek a suitable service and negotiate with it.

---

shifting the focus from providing software to describing and delivering a service moves the focus away from the constraints that traditional software construction, use, and ownership models impose. Hence, our service-based model configures, executes, and disengages one or more services to meet a specific set of requirements[3]—a vision of instant service consistent with the widely accepted definition: "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production."[4]

Figure 1a shows an abstract interpretation of what the current literature widely refers to as a service model: A fixed set of applications sit on top of a service transport layer that uses technologies such as Microsoft's .NET or Sun's J2EE platform, along with XML-based enveloping and message formats such as the simple object access protocol (SOAP). Figure 1b shows our vision of what a service model

Figure 1. Service models. (a) The current supply-led service model provides only a predetermined range of services from a remote server. (b) The proposed demand-led service model has a service integration layer inserted above the transport layer.

could be if developers insert a further service integration layer above the transport layer.

The model in Figure 1a is *supply-led* because it supports applications that can provide only a predetermined range of services from a remote server. The model in Figure 1b is *demand-led* because applications can be constructed from smaller component services and bound dynamically as needed. Providing this capability will require support from high-bandwidth information networking, which

we expect emerging grid technologies to provide and enhance.

SERVICE INTEGRATION LAYER

The key to the quite radical difference between the two models represented in Figure 1 lies in the functionality that results from combining the facilities that the service integration layer provides with those from the component services that make up an application. This layer employs software technology to support a set of concepts closely related to business and supply models. The service integration layer incorporates four key service-oriented functions.

Service description

This function matches client needs to appropriate and available services. Essentially, the service description provides the means of mapping between the provider's description of its offerings and the client's description of its needs. The form used should accommodate descriptions of functionality, interfaces, and nonfunctional characteristics and constraints such as quality of service and cost. It should also describe the parameters within which both the service provider and client will negotiate.

Service discovery

Clients use service discovery to locate appropriate services, according to their requirements and selection criteria. Using this process, a client identifies those potential service providers whose offerings meet its functional needs and who are prepared to negotiate within some acceptable bounds. Discovery can involve the recursive use of other services, including brokers, and will result in a list of candidate services and providers. *Service negotiation* involves the interaction between a client and one or more of the service providers identified through the discovery process or already known to the client. This negotiation aims to achieve agreement on the terms and conditions for supplying a service.

Service delivery

This function consists of three steps. Invocation is the calling-for step, during which the client requests the provider to supply the specified service according to the agreed terms and conditions. Next, to validate the invocation, in the provision step the service provider must supply the agreed-upon service within the period agreed to in the supply contract. Finally, where the contract has unspecified

bounds of provision, or when the bounds are reached, the *suspension* step establishes the point at which the client no longer needs the provider to supply the service.

Service composition

In its most direct form, know-how—expressed in terms of rules—drives service composition so that a service provider can compose its service from lower-level services. However, such knowledge is sufficient only for constructing those services for which rules already exist. In the longer term, we seek to devise a suitable mechanism for creating new forms of service on demand. Nothing in our model prevents this, but creating the means of automatically providing entirely new services will clearly only be practical once the other elements of the service integration layer have been fully developed.

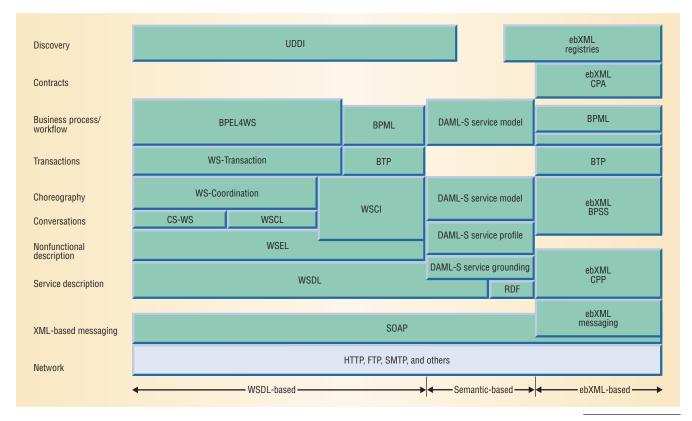## CURRENT SERVICE-RELATED PROTOCOLS

The current research literature frequently uses the service-model concept to describe Web service technologies such as Microsoft's .NET platform. Although the Web services paradigm is fairly consistent with our vision of SaaS, creating a true service-oriented marketplace requires further developments.

Since the introduction of Web services, three XML-based protocols have become de facto standards. These three protocols have become so widespread that the term Web services has become synonymous with them:

- SOAP provides a message format for communicating with and invoking Web services;
- the Web Services Description Language (WSDL) describes how to access Web services; and
- universal description, discovery and integration (UDDI) provides a registry that clients can use to discover available services.

These three protocols are adequate for simple Web services requiring a remote-procedure-call style of communication. For more complex Web services that consist of several services, other XML-based specifications provide functions at higher or intermediate layers in a stack of protocols.

When developing complex Web services, the lack of a universally accepted protocol that provides all the functionality required at each layer can cause problems. Adding to this confusion is the lack of an overall definition for the actual layers such a stack requires. The many standards organizations and companies involved all have different visions of

Figure 2. Proposed Web services stack framework. Separated into three vertical sections, the protocols use or extend WSDL, have roots in the semantic Web's resource description framework, and the DARPA Agent Markup Language for Services (DAML-S), and include ebXML specifications.

the layers and protocols that make up the Web services architecture.

IBM produced one of the stack's original definitions in its Web Services Conceptual Architecture document.[5] It included the three de facto standards at the XML-based messaging, service implementation, and description and discovery layers, along with a service flow layer that incorporated IBM's Web Services Flow Language.[6] However, the latter has now been combined with Microsoft's XLANG protocol (www.gotdotnet.com/team/xml_wsspecs/ xlang-c/default.htm) to produce a new set of protocols: the Business Process Execution Language for Web Services (BPEL4WS; www-106.ibm.com/ developerworks/webservices/library/ws-bpel/). The W3C Web Services Architecture group also is working on its own stack version to standardize the required layers,[7] again emphasizing the three basic protocols.

Few of the available stacks include any detail on the semantic Web protocols or the more business-oriented Electronic Business using Extensible Markup Language (ebXML). As a result, which technologies to use at each level—and even which of the available technologies are compatible— remains unclear. To this end, we propose an updated Web services stack framework that places the currently available initiatives in context.

The stack framework shown in Figure 2 consists of several open-systems-interconnection-type layers, with each level using the services of the levels below it.

- *Network*. This is the underlying transport protocol layer.
- *XML-based messaging*. XML is the message format for communicating documents and procedure calls. This layer can, for example, use SOAP with any underlying transport protocols in the network layer. This layer decouples messaging from the physical transport protocol so that messages can concentrate on describing the service semantics. The Electronic Business XML solution, ebXML Messaging Specification (ebMS), builds on SOAP by using its header specification extensibility to include authentication and contextual information.
- *Service description*. This layer provides the functional description of a Web service in terms of its interface and implementation. The majority of description languages at this layer utilize the XML Schema language for expressing data-type information.
- *Nonfunctional description*. Protocols at this layer describe a service in terms of its less technical features, such as quality of service, cost, geographic location, number of retries, and legal factors.
- Conversations. In this context, a conversation refers to the external view of the messages a Web service is receiving and sending. This layer therefore describes the correct data types and sequence of messages or documents a Web service is exchanging.
- *Choreography*. Whereas all the previous lay-

ers are largely concerned with describing a single Web service, the choreography layer coordinates several Web services into a pattern to provide an overall outcome. For example, protocols in this layer would specify the order in which the methods—or *operations* in WSDL terminology—of each Web service must be invoked.

- *Transactions*. The protocols in this layer facilitate monitoring transactions between Web services. When services themselves consist of other services, numerous points of failure become possible. The transactions layer describes how to achieve this composition in an atomic way, so that, for example, the entire process either completes successfully or rolls back.

- *Business process and workflow*. The protocols in this layer describe how to actually compose a higher-level service from several other Web services, through descriptions of the control and data flows involved in the process. It differs from the choreography layer by providing internal details of the composition to supply an executable business process.

- *Contracts*. This layer outlines the format of the machine-readable contracts necessary to automate service-based electronic business. The contract outlines the transaction's terms and conditions and finalizes any negotiable parameters, such as cost and acceptable time to delivery.

- *Discovery*. Providers use this layer to publish details of their Web services so that clients can then search and discover any that meet their needs.

We also separate our stack into three vertical sections. The first section includes protocols that use or extend WSDL. The second includes protocols that have roots in the semantic Web: the resource description framework (RDF) and the DARPA Agent Markup Language for Services (DAML-S).[8] As Figure 2 shows, WSDL crosses the boundary into the semantic-based section in our stack. This is not because WSDL is semantic-based, but because DAML-S builds upon WSDL for its Service Grounding specification. The ebXML specifications comprise the third vertical layer in our stack. Although it is independent of Web services, ebXML offers much the same functionality as the other sections.

Also, several protocols relate to security within the Web services paradigm, including enhance-ments added to SOAP to provide secure messaging capabilities such as those that WS-Security defines. However, because it concentrates instead on other areas, our current model does not include security.

REALIZING THE SERVICE INTEGRATION LAYER

Figure 2 reveals several significant gaps in the current Web services stack.

Service description

Current description methods suffer from a severe limitation: Although they provide the technical information a client requires to invoke a service, they cannot describe the function the service provides semantically. They also lack descriptions of service delivery's negotiable aspects. For example, although it is the de facto standard Web services language, WSDL describes a service in terms only of its acceptable data types, methods, message format, transport protocol, and end-point uniform resource identifier (URI).

In the ebXML section of our stack framework, the Collaboration Protocol Profile specification deals with service description. Although it includes more details about the service provider and error-handling scenarios than WSDL, the CPP focuses largely on the transaction's technical aspects. In the WSDL-based section of our framework, IBM's Web Services Endpoint Language[6] is the only protocol explicitly designed to describe the nonfunctional, negotiable elements of Web services. WSEL, however, remains a work in progress.

DAML-S is the only available description method designed specifically for describing the functional and nonfunctional aspects of services, including details of what they actually do. A client can use the specification's service profile to describe its requirements, and the service provider can use the service profile to describe its capabilities, including nonfunctional parameters, in a semantically rich, ontology-based format. However, while closest to our requirements, DAML-S has not yet reached a final release. Thus, the exact details of what the service profile will include have yet to be finalized, and it also currently lacks an extensive selection of usable, standardized ontologies. Support for the language within current tools is also fairly limited.

Service discovery

UDDI is the de facto standard for discovery in the Web services environment. For our purposes, UDDI's key limitation lies in its inability to allow semantic descriptions. This limits searching to key-

words, such as the name of the service provider or the service itself, the service's location, or the business classification. Although the ebXML registry specification offers richer searching capabilities in the form of SQL and XML filter queries, it does not allow semantic searching. Therefore, a client cannot use either of the available registry specifications to search for a service based on its functionality, which limits the current model's dynamic discovery capabilities.

Clients can use the DAML-S service profile to make requests for services, and providers can use it to describe their services semantically by the functionality they provide. Because the language is ontology-based, its inferential capabilities should allow matching requests to service descriptions. However, this requires a registry capable of performing true semantic matching, which UDDI currently does not do. To this end, researchers have been using an algorithm from previous research that matches requests to advertisements according to their semantics[9] to determine how to extend UDDI with DAML-S.

Service negotiation

The SaaS model requires that, upon discovering a suitable service, the client and provider must negotiate the service's delivery terms and conditions automatically. Although several protocols throughout the stack framework include descriptions of negotiable parameters—including ebXML's CPP, DAML-S, and WSEL—none allow for fully automated negotiation. The model also needs electronic contracts to seal any negotiations that take place. Figure 2 shows that, among the current approaches, only ebXML includes such contracts.

The Collaboration Protocol Agreement primarily defines the common protocols and capabilities of only two parties. Formed from the intersection of the two parties' CPP documents, the CPA uses XML to define properties such as the contract's duration and the transactions' agreed security features. Formulating a CPA is intended to be a manual process.[10]

Service delivery

Our model identifies three steps of primary importance to service delivery. Current technologies cover a Web service's basic invocation and provision, but they do not support either monitoring whether the service is supplied within the agreed terms and conditions or suspending the provision, if necessary. Doing so would require an electronic contract, which only ebXML includes. Services can use ebXML's CPA document to monitor the trans-

actions and terminate the process if this contract is broken. However, in relation to our model, this document does not detail any legal or nonfunctional parameters such as cost or quality of service.

Several other protocols in the stack, particularly the Web Services Choreography Interface and WSEL, also include elements that touch upon service monitoring and suspension. WSCI lets the developer specify how a service will react in exceptional circumstances, while both WSCI and WSEL can detail time-out periods. Thus, developers could use these elements to monitor the transaction at a basic level.

Also, true dynamic binding and invocation relies heavily on the richness of the service description. This description must be machine readable and semantically rich to enable the requesting service to automatically decipher all the requirements of use. This capability is not yet fully in place.

Service composition

The automatic composition of Web services requires suitable protocols at the *conversations*, *choreography*, *workflow*, and *transactions* layers. As Figure 2 shows, several protocol combinations are available at each of these layers within the WSDL-based section:

- HP's Web Services Conversation Language (WSCL) and IBM's recent Conversation-Support for Web Services specification (CS-WS) both cover the conversation layer,[11]
- the WS-Coordination protocol and WSCI both cover the choreography that links each of the collaborating Web services together, and
- the Business Transaction Protocol (BTP) and WS-Transaction both cover monitoring and handling long-running business transactions.

Developers can use either the Business Process Modeling Language or the Business Process Execution Language for Web Services to model the actual control and data flows within the composition. BPEL4WS, which is likely to become more widely adopted, can be distinguished from other protocols in the layer because it includes both an abstract XML description and an executable language. The actual BPEL4WS specification also encompasses the WS-Coordination and WS-Transaction protocols, thus ensuring compatibility among the three layers.

The Business Process Specification Schema (BPSS) provides an ebXML alternative. When combined

> **The SaaS model requires that the client and provider negotiate the service's delivery terms and conditions automatically.**

with the compatible BPML and BTP specifications, it can also describe the internal details of workflows and long-running transactions. However, the BPSS is designed to model only a transaction between two parties, not a complex Web services composition.

DAML-S covers many of the compositional layers with the service profile and service model specifications. However, the current release lacks support for long-running transactions. We expect that a future specification will include such support, along with a control model that describes a process in terms of its state, allowing automated monitoring.

Although many available technologies support service composition, they require that the developer knows, at design time, the details of the services to be used. In the SaaS model, services will be dynamically composable when needed, through binding of several other, lower-level services. Indeed, a true service-oriented model will automatically compose a new higher-level service from a client's description of the sequence of tasks to be performed. The service could achieve this by searching for and dynamically binding to lower-level services that perform each task.

Although many of the protocols necessary to achieve true service provision are either available or under development, significant gaps remain. Perhaps not surprisingly, these protocols address the less technical aspects of service delivery and thus represent research challenges that have strong interdisciplinary elements. ■

References

1. O.P. Brereton and D. Budgen, "Component-Based Systems: A Classification of Issues," *Computer,* Nov. 2000, pp. 54-62.
2. K.H. Bennett et al., "An Architectural Model for Service-Based Software with Ultra-Rapid Evolution," *Proc. Int'l Conf. Software Maintenance* (ICSM 2001), IEEE CS Press, 2001, pp. 292-300.
3. O.P. Brereton et al., "The Future of Software," *Comm. ACM*, Dec. 1999, pp. 78-84.
4. C. Lovelock, S. Vandermerwe, and B. Lewis, *Services Marketing*, Prentice-Hall, 1996.
5. H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0)," 2001; www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.
6. F. Leymann, "Web Services Flow Language (WSFL) 1.0," 2001; www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
7. D. Booth et al., "Web Services Architecture: W3C Working Draft," 14 May 2003; www.w3.org/TR/ws-arch/.
8. A. Ankolekar et al., "DAML-S: Web Services Description for the Semantic Web," *Proc. 1st Int'l Semantic Web Conf.* (ISWC 2002), Springer-Verlag, 2002, pp. 348-363.
9. M. Paolucci et al., "Importing the Semantic Web in UDDI," *Proc. Web Services, E-Business, and Semantic Web Workshop* (CAiSE 2002), Springer-Verlag, 2002, pp. 225-236.
10. ebXML Trading-Partners Team, "Collaboration-Protocol Profile and Agreement Specification," v. 1.0, 2001; www.ebxml.org/specs/ebCCP.pdf.
11. J.E. Hanson, P. Nandi, and S. Kumaran, "Conversation Support for Business Process Integration," *Proc. 6th IEEE Int'l Enterprise Distributed Object Computing Conf.* (EDOC 2002), IEEE CS Press, 2002, pp. 65-75.

Mark Turner is a research assistant and PhD student in the Department of Computer Science at Keele University, UK. His research interests focus on service-based software engineering, in particular access control, description languages, and Web services substitution. Turner received an MSc in information technology from Keele University. Contact him at m.turner@cs.keele.ac.uk.

David Budgen is a professor of software engineering at Keele University. His research interests include design, measurement, and empirical evaluation practices for software-based systems. Budgen received a PhD in theoretical physics from the University of Durham. Contact him at d.budgen@cs.keele.ac.uk.

Pearl Brereton is a professor in the Department of Computer Science at Keele University. Her research focuses on component-based and service-based software engineering. Brereton received a PhD in numerical analysis from Keele University. Contact her at o.p.brereton@cs.keele.ac.uk.