Turning Software into a Service

Mark Turner, David Budgen, Pearl Brereton
Keele University,
Staffordshire,
ST5-5BG
m.turner@cs.keele.ac.uk

1. Introduction

In 1999 the Pennine Group (http://www.service-oriented.com), a consortium of software engineering researchers from the University of Durham, Keele University and UMIST, put forward the view that the future of software lay not in developing new architectural styles based upon 'constructional' forms, such as objects or components, but in taking a radically different view of the way that its functionality was delivered to the user (Brereton, 1999). In subsequent papers we have explored this concept further and have illustrated it by means of a number of small-scale experiments (Bennett *et al.*, 2001). This idea, which we termed *Software as a Service* (SaaS), is one of a demand-led software market in which services are assembled and provided as and when needed to address a particular requirement. In this paper we illustrate how we believe the SaaS vision compares with current thinking about software development and delivery and discuss whether the currently available technologies are sufficient to support our ideas. In particular, we examine the initiatives in the Web services and electronic business communication communities and develop a *stack framework* through which to compare a number of available protocols. We then go on to examine the gaps that this exposes in the current Web services protocols, and in so doing highlight where further research is needed in order to effectively realise the SaaS vision.

1.1 The Service Concept

The basic long-term vision of SaaS is centred around separating software *possession* and *ownership* from its *use*. The Pennine Group believe that by delivering the functionality of software as a set of services that can be configured and bound at the time of delivery, many of the present limitations constraining its use, deployment and evolution can be overcome. Such a model would open up new markets, both for relatively small-scale providers of specialist services as well as for larger organisations that provide more general services. In addition, service provision may include the dynamic creation and development of entirely new services that make use of existing ones.

One feature of this approach is that as a business and its context change, so the set of services that it uses will evolve without any user intervention. Also, note that the key 'know-how' involved is not one of *who* provides services, necessary though that knowledge is, but more one of knowing *what* service is required at any particular point and negotiating suitable terms for its use. Selecting and binding the means of providing an appropriate service can therefore be performed dynamically on demand (*ultra-late binding*).

1.2 What distinguishes SaaS from other 'Service forms'?

Despite advances in the technologies of programming languages and development environments, the basic paradigm used for constructing and maintaining software has altered very little since the 1960's. Software is still largely constructed by employing some variant of the *edit-compile-link* cycle to generate an executable 'binary image' from a source that is described using some form of 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 are not so very different from those traditionally employed for constructing software, and are just as error-prone!

To achieve the above vision, the Pennine Group is developing a radically different paradigm. Our view is that the role of software is to deliver a *service* and that by shifting the focus to describing and delivering that service rather than providing software, we can move away from the constraints imposed by the 'traditional' models of software construction, use and ownership. Hence our **service-based model** is one in which one or more services are configured to meet a specific set of requirements at a point in time, executed and disengaged (Brereton *et al.*, 1999) - a vision of instant service that is 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" (Lovelock *et al.*, 1996)

Figure 1 Current Service Model

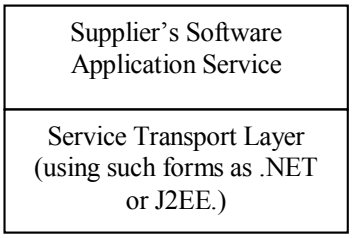

Figure 2 Proposed Service Model

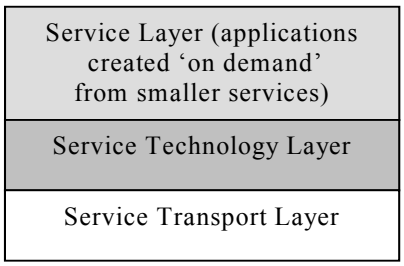

Figure 1 shows an abstract interpretation of what is widely meant by a 'service model' in the current literature[1]. In this, a fixed set of applications sit on top of a **Service Transport Layer** that makes use of such technologies as Microsoft's .NET or Sun's J2EE platform, along with XML based enveloping and message formats such as SOAP (http://www.w3.org/TR/SOAP/). Figure 2 shows our vision of what a 'service model' *could* be, if a further **Service Technology Layer** is inserted above the Transport layer. The model of Figure 1 is essentially a *supply-led* one, concerned with supporting applications which can only provide a pre-determined range of services from a remote server; while that of Figure 2 is a *demand-led* one, that will enable applications to be constructed from smaller 'component services' and bound dynamically as needed. This will need to be supported by high bandwidth networking of information, which we expect will be provided and enhanced further through the emerging Grid technologies.

2. The Service Technology Layer

The key to the quite radical difference between the ideas represented by Figures 1 and 2 lies in the functionality that results from combining the facilities provided by the Service Technology Layer with those from the component services that make up an application. In this section we further examine the role of this layer, and identify the functions that it needs to provide in order to operate as a true SaaS environment.

Although we have used the term Service Technology Layer, we should note that the role that we assign to it is one that involves a wide interpretation of the term 'technology'. Indeed, it employs software technology to support a set of concepts that are more closely related to business and supply models than to technological ones. Within our model, we have identified the following as being key service-oriented functions.

Service description is needed as a basis for matching client needs to appropriate and available services. Essentially, the service description provides the means of mapping between the provider's description of their offerings and the client's description of their needs. The form used should accommodate description of functionality, interfaces and non-functional characteristics/constraints, such as quality of service and cost. It should also describe the parameters within which both the service provider and client are willing to negotiate.

Service discovery encompasses the method by which clients will locate appropriate services, according to their requirements and selection criteria. It is the process by which a client identifies those potential service providers whose offerings meet their functional needs and who are prepared to negotiate within some acceptable bounds. Discovery may 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. It has the aim of agreeing the terms and conditions for the supply of a service.

Service delivery is comprised of three steps: *invocation*, *provision* and *suspension*. *Invocation* is the 'calling for' step where the client requests the provider to supply the specified service within the agreed terms and conditions. For this step to be valid, the service provider must supply the agreed service in a manner, and within the time frame, agreed in the contract of supply, which is the *provision* step. Finally, where the bounds of the

[1] As an example of this, the *Financial Times* of 1 May 2002 devoted an 8-page supplement to this concept, which is really little more than a return to the old 'bureau' concept, but delivering fixed services on-line instead of off-line!

provision are not specified in the contract, or when the bounds are reached, the *suspension* step establishes the point at which the client no longer requires the supply of a service.

Service composition in its most direct form is driven by the 'know-how' (expressed in terms of rules) that enables a service provider to compose its service from lower level services. However, such knowledge is only sufficient for constructing those services for which rules already exist. A longer-term research goal is 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 technology layer have been fully developed.

3. Current Service-related Protocols

As we previously observed, the concept of a 'service model' is widely adopted in the current literature to describe Web service technologies such as Microsoft's .NET platform. Whilst the Web services paradigm is fairly consistent with our vision of SaaS, we feel that some further developments are needed before a true service-oriented marketplace is feasible. In this section we examine some current Web services initiatives and propose a stack framework that illustrates the functions that each provides and how we believe each of the languages or protocols[2] interrelate. We then discuss how this stack framework maps onto the requirements of our Service Technology Layer.

Since the introduction of Web services, three XML based protocols have become de-facto standards. In fact, they have become so widespread that the term Web services has become synonymous with these three protocols. They are: **SOAP**, or Simple Object Access Protocol, providing a message format for communicating with and invoking Web services; **WSDL**, or Web Service Description Language (http://www.w3.org/TR/wsdl.html), describing how to access them; and **UDDI**, or Universal Description, Discovery and Integration (http://www.uddi.org), providing a registry that clients can use to discover available services.

These three protocols are adequate for simple Web services requiring remote procedure call style communication. For more complex Web services, perhaps being composed of a number of other services, other XML-based protocols have been developed to provide functions at higher or intermediate layers in the stack. One of the problems in developing complex Web services is that there is no universally accepted protocol that provides all of the functionality required at each layer. To add to the confusion, there is no overall definition of the actual layers that are required in the stack. The numerous standards organisations and companies involved all have different visions of the layers and protocols making up the Web services architecture. IBM produced one of the original definitions of a stack in their Web Services Conceptual Architecture document (Kreger, 2001). This included the three de-facto standards mentioned previously, along with a 'Service Flow' layer that incorporated IBM's **Web Services Flow Language** (WSFL) (Leymann, 2001). However, the latter has now been combined with Microsoft's **XLANG** protocol to produce a new set of protocols termed the **Business Process Execution Language for Web Services** (BPEL4WS) (http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/). The W3C Web Service Architecture group are also working on a version of their own stack to standardise the required layers (Champion et al., 2002), again placing particular emphasis on the three basic protocols. Very few of the available stacks include any detail on the Semantic Web protocols or the more business-oriented Electronic Business XML (ebXML) (http://www.ebxml.org). As a result, it is unclear as to which technologies to use at each level and even which of those available are compatible. To this end, we propose an updated Web Services stack framework that attempts to place the initiatives available at the time of writing in some form of context. This is illustrated in Figure 3.

[2] The terms 'language' and 'protocol' can, in the vast majority of cases, be used interchangeably when referring to XML-based Web services initiatives. From this point on, we will use the term 'protocol' for conciseness.

Figure 3 Proposed Web Services Stack Framework
(Please see separate PDF file, Figure3.pdf)

The stack framework shown in Figure 3 consists of a number of OSI-type layers, with each level using the services of the levels below it. The layers are:

- *Network*. This layer consists of the underlying transport protocol.
- *XML-based messaging*. This layer encompasses the use of XML as a message format to communicate documents and procedure calls. SOAP, for example, can be used with any of the underlying transport protocols in the Network layer. This de-coupling of messaging from physical transport protocol means that messages can concentrate on describing the service semantics.
- *Service Description*. This is concerned with the functional description of a Web service, in terms of its interface and implementation. It is this layer, and those above it, that we will concentrate on throughout the remainder of this paper.
- *Non-functional 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, the term conversation refers to the external view of messages being received by and sent from a Web service. This layer therefore describes the correct sequence of messages (or documents) to be exchanged with a Web service.
- *Choreography*. All of the previous layers have largely been concerned with describing a *single* Web service, whereas the choreography layer describes how the methods (or *Operations* in WSDL terminology) of several Web services can be coordinated to provide an overall outcome. For example, protocols in this layer would specify the order in which the methods of each Web service would need to be invoked.
- *Transactions*. This layer includes protocols that facilitate the monitoring of transactions between Web services. When services are themselves composed from other services, there are numerous points of failure. The transactions layer would describe how to achieve this composition in an atomic way, so that, for example, the whole process either completes successfully or is rolled back.
- *Business Process/Workflow*. Protocols in this layer describe how to actually compose a higher-level service from a number of other Web services, through descriptions of the control and data flows involved in the process.
- *Contracts*. This layer outlines the format of machine-readable contracts that are necessary to automate service-based electronic business. The contract would outline the terms and conditions of the transaction and finalise any negotiable parameters, such as cost and acceptable time to delivery.
- *Discovery*. This layer allows providers to publish details of their Web services so that clients can then search and discover any that meet their need.

We also separate our stack into three vertical sections. The first we have termed 'WSDL-based' and includes those protocols that use or extend WSDL in some way. Secondly we include those protocols that have their roots in the semantic Web: the Resource Description Framework (RDF) (http://www.w3.org/RDF/) and the DARPA Agent Markup Language for Services (DAML-S) (DAML Services Coalition, 2002). Examining Figure 3, it can be seen that in our stack WSDL crosses the boundary into the 'Semantic based' section. This is not because WSDL is in any way semantic based, but because DAML-S builds upon WSDL for its Service Grounding specification. Finally, the ebXML specifications are included in our stack. ebXML, although independent of Web services, offers much of the same functionality as the other sections.

There are also a number of protocols relating to security within the Web services paradigm, including enhancements added to SOAP to provide secure messaging capabilities, such as that defined by WS-Security (http://www-106.ibm.com/developerworks/library/ws-secure/). We do not include security in our current model, as we aim solely to concentrate on the areas identified in Section 2.

4. Realising the Service Technology Layer

As can be seen from Figure 3, in our view there are a number of significant gaps in the current Web services stack. In the rest of this section we examine how the protocols at each layer can contribute to the five key service-oriented requirements of our Service Technology Layer. In doing so, we identify what needs to be addressed in the current Web services model in order to begin implementing the ideas inherent in SaaS.

Service Description
The main problem with current description methods is that whilst they provide the technical information required for a client to invoke a service, they lack the ability to describe semantically the function that the service provides. They also lack descriptions of the negotiable aspects of service delivery. This is true of the de-facto standard description language in the Web services world, **WSDL**, and also for **ebXML's Collaboration Protocol Profile** (CPP) specification. The latter, whilst including more details about the service provider and error handling scenarios, is still based largely around the technical aspects of the transaction. In the 'WSDL-based' section of the stack framework shown in Figure 3, the only protocol designed to describe the non-functional, negotiable elements of Web services is IBM's **Web Services Endpoint Language (WSEL)** (Leymann, 2001). However, this has yet to be developed beyond the status of work-in-progress.

DAML-S is the only description method available that is designed specifically to allow service providers to describe the functional and non-functional aspects of their services, including details of what they actually *do*. The Service Profile aspect of the specification allows the client to describe their requirements and the service provider to describe their capabilities, including non-functional parameters, in a semantic rich ontology-based format. However, whilst closest to our requirements, at the time of writing DAML-S is not in a final release and as such is not widely supported.

Service Discovery
UDDI is the de-facto standard for discovery in the Web services environment. The key problem for our purpose is that UDDI does not allow for semantic descriptions. Therefore, searching is limited to keywords, such as the name of the service provider or service itself, the location of the service, or the business classification. Whilst the **ebXML registry specification** (http://www.ebxml.org/specs/ebrs2.pdf) offers richer searching capabilities in the form of SQL and XML filter queries, it lacks the ability to allow semantic searching. Therefore, neither of the available registry specifications allows a client to search for a service based on its functionality, thus limiting the dynamic discovery capabilities of the current model.

The DAML-S Service Profile ontology allows clients to make requests for services and enables providers to describe their services semantically on the basis of the functionality they provide. As the language is based around ontologies, it should therefore be possible to use the inferential capabilities of the language to match requests to service descriptions. However, this requires a registry that is capable of performing true semantic matching, which UDDI currently fails to do. To this end, research has been conducted into how to extend UDDI with DAML-S**,** utilising an algorithm from previous research that enabled the matching of requests to advertisements according to their semantics (Paolucci et al., 2002).

Service Negotiation
The SaaS model requires that when a suitable service is discovered, the client and provider will need to negotiate the terms and conditions automatically for the delivery of that service. While a number of the protocols throughout the stack framework include descriptions of negotiable parameters (including ebXML's Collaboration Protocol Profile, DAML-S and WSEL), none allow for fully automated negotiation.

Electronic *contracts* are also needed to seal any negotiations that take place. Figure 3 illustrates that, amongst the current approaches, only ebXML includes electronic contracts. The **Collaboration Protocol Agreement** (CPA) document is designed primarily to define the common protocols and capabilities of only two parties. It is formed from the intersection of the two party's Collaboration Protocol Profile documents and defines, in XML format, such properties as the duration of the contract and the agreed security features of the transactions. Information on how to formulate a CPA is available (ebXML Trading-Partners Team, 2001) but this is intended to be a manual process.

Service Delivery
Our model identifies three steps that are of prime importance to service delivery: invocation, provision and suspension. The basic invocation and provision of a Web service are well covered by the current technologies. However, the ability to monitor whether the service is supplied *within the agreed terms and conditions*, and the ability to *suspend* the provision if necessary are not. For these to be viable there would need to be an electronic contract and only ebXML includes this feature. Services can use ebXML's CPA document to monitor the transactions and terminate the process if this contract is broken. However, in relation to our model it does not detail any legal or non-functional parameters, such as cost or quality of service.

A number of other protocols in the stack, in particular the **Web Services Choreography Interface** (WSCI) (www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf) and the Web Services Endpoint Language, also include elements that touch upon the idea of service monitoring and suspension. WSCI allows the developer to specify how their service will react in exceptional circumstances, and both WSCI and WSEL allow timeout

periods to be detailed. Thus, these elements could be used to monitor the transaction at a basic level.

Service Composition
Achieving the automatic composition of Web services will require suitable protocols to be available at the *conversations, choreography*, *workflow* and *transactions* layers. As can be seen in Figure 3, there are several combinations of protocols available at each of these layers within the 'WSDL-based' section. The conversations layer is covered by both HP's **Web Services Conversation Language** (WSCL) (http://www.w3.org/TR/wscl10/) and IBM's recent **Conversation-Support for Web Services specification** (CS-WS) (Hanson, Nandi, Kumaran, 2002). Choreography is used to link each of the collaborating Web services together and is covered by the **WS-Coordination** protocol, as well as **WSCI**.

The monitoring and handling of long-running business transactions is covered by both the **Business Transaction Protocol** (BTP) (http://www.oasis-open.org /committees/business-transactions/documents/specification/2002-06-03.BTP_cttee_spec_1.0.pdf) and **WS-Transaction**. To model the actual control and data flows within the composition, **BPEL4WS** or the **Business Process Modeling Language** (BPML) (http://www.bpmi.org/bpml.esp) can be used. BPEL4WS is looking likely to become the most widely adopted and is distinguishable from others in the layer through the inclusion of 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 between the three layers.

The ebXML alternative is provided by the **Business Process Specification Schema** (BPSS) specification. When combined with the compatible BPML and BTP specifications, it can also describe, respectively, the internal details of workflows and long running transactions. However, the BPSS is only designed to model the transaction between two parties as oppose to a complex Web services composition.

DAML-S covers many of the compositional layers with the *Service Profile* and *Service Model* specifications. At the time of writing, the specification has only reached version 0.7 and there is still more to be implemented before a full release, in particular features are needed to support long running transactions. It is envisaged that a future specification will include a 'Process Control' model that will describe a process in terms of its state, allowing for automated monitoring.

5. Conclusions

This paper has elaborated on our vision of a radically new approach to using software, where the notion of software 'ownership' is discarded in favour of service acquisition, and examined whether the vision can be achieved using currently-available Web services protocols. We have described the basic components of the vision, and have argued that even though the underpinning technologies necessary for realising the vision are largely in place there are still a number of areas that will need further research before the ideas can be fully implemented. We have illustrated this through use of a comparative stack (Figure 3) that highlights the significant gaps.

The gaps we have identified in the current Web services technologies will, along with the ongoing research into the inter-disciplinary and market driven aspects, provide areas of future research for The Pennine Group.
The first area we identified concerned the lack of flexible, semantic based searching in the current discovery models. In our model, clients and other services need to be able to discover services dynamically on the basis of their capabilities and bind to them at run-time. The current service registries do not fully allow for this. The DAML-S language is designed to allow capability matching but at the time of writing it is not in a final release and is not supported directly by either the UDDI or ebXML registry models.

Secondly, once a service has been discovered, the terms and conditions of a subsequent transaction will need to be negotiated. Again this process should occur automatically at run-time. However, current Web services protocols do not include support for automatic negotiation, especially for more complex issues such as legal clauses. This problem is exaggerated further by the fact that the current protocols place very little emphasis on the contextual aspects of a transaction, such as *cost* or *quality of service*. This means that many of the negotiable elements are not described and so it is not possible for a client to choose a service on the basis of its non-functional elements, or to choose a service and then enter the negotiation process if the current terms are unsatisfactory in some way.

Finally, although there are many technologies available that provide support for service composition, they require that the developer knows, at design time, the details of the services that are to be used. In the SaaS model, services should be dynamically composable at the time of need, through binding of a number of other, lower-level, services. Indeed, a client should be able to describe the sequence of tasks to be performed, and a true

service-oriented model would automatically compose a new higher-level service from this description. This would be achieved through searching for, and dynamically binding to, 'lower-level' services that perform each task.

From the above we therefore conclude that, while many of the protocols necessary to achieve 'true' service provision are either available or under development, there are still significant gaps. Perhaps not surprisingly, these are concerned with the less 'technical' aspects of service delivery and so represent research challenges that have strong interdisciplinary elements.

References

Bennett KH, Munro M, Gold N, Layzell PJ, Budgen D & Brereton OP (2001) *An Architectural Model for Service-Based Software with Ultra-Rapid Evolution*. In *Proceedings of ICSM'01*, Florence, November 2001, IEEE Computer Society Press, 292-300

Brereton OP, Budgen D, Bennett KH, Munro M, Layzell PJ, Macaulay L, Griffiths D & Stannett C (1999) *The Future of Software*, Comm. ACM, **42**(12), 78-84

Brereton OP & Budgen D (2000) *Component-Based Systems: A Classification of Issues*, IEEE Computer, **33**(11), 54-62

Champion, M, Ferris, C, Newcomer, E. & Orchard, D (2002), *Web Services Architecture – Editors' Copy* [Online], Available:http://www.w3.org/2002/ws/arch/2/08/wd-wsa-arch-20020821.html [2002, Oct. 15]

DAML Services Coalition (2002), *DAML-S: Web services Description for the Semantic Web*, In: Proceedings of the 1st International Semantic Web Conference (ISWC)

ebXML Trading-Partners Team (2001), *Collaboration-Protocol Profile and Agreement Specification Version 1.0* [Online], Available: http://www.ebxml.org/specs/ebCCP.pdf [2002, Sept. 12]

Hanson, J.E, Nandi, P & Kumaran, S (2002), *Conversation Support for Business Process Integration*, In: Proceedings of the 6th IEEE International Enterprise Distributed Object Computing Conference EDOC 2002-09-17

Kreger, H (2001), *Web Services Conceptual Architecture (WSCA 1.0)* [Online], Available: http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf [2002, Oct. 15]

Leymann, F (2001), *Web Services Flow Language (WSFL) 1.0*, [Online], Available: http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf [2002, Sept 2]

Lovelock C, Vandermerwe S, & Lewis B (1996) *Services Marketing*, Prentice Hall Europe

Paolucci, M, Kawamura, T, Payne, T.R, & Sycara, K (2002), *Importing the Semantic Web in UDDI*, Forthcoming in Proceedings of Web Services, E-business and Semantic Web Workshop

(SIDEBAR) An example scenario to illustrate the concept of *Software as a Service*

The following scenario offers a simple illustration of the ideas inherent in Software as a Service, set within the context of a much larger example.

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

Alice's purchasing service uses other services to handle such tasks as translation, legal and financial negotiations, financing, and currency transfer. So provision of her service involves specifying the terms, conditions and form of service provision, together with the 'rules' describing how other services will be employed to provide this (the 'know-how' element of her service). A complete scenario that describes this would be much too long, but a small fragment (concerning legal documents) might look like the following:

Alice's purchasing service urgently needs to have a legal document translated from Spanish to English.

- It seeks a translation service and, after negotiation on the terms and conditions, it selects the cheapest from the four available (*scribe*).
- *scribe* (which is a broker, able to evaluate services, but not actually providing them) seeks a Spanish translation service that is geared to handling "legal" documents and from the three on offer, it selects the one that offers an immediate service (*es-trans*), based upon previous history of use and satisfactory delivery.
- *es-trans* provides the service that Alice's service requires.

In order to further explain the SaaS vision, we now relate this example back to the five key service-oriented functions of our Service Technology Layer. Firstly, in terms of service description, Alice would need to be able to describe her general requirement to have a document translated, and more specific details including the fact that it is a *legal* document, and the languages involved. It also encompasses the way that a service provider such as *scribe* or *es-trans* can describe the services that they can supply (either directly, or through use of other providers) and the parameters within which they are willing to negotiate a service contract.
There are two instances of service discovery in this example. The first is Alice seeking the translation service, and the second is the *scribe* broker seeking a service that can deliver in the required time. Likewise, two stages of negotiation occur in the example. Alice's service will negotiate with *scribe* (but because she has a frequent need for translation services in general, this negotiation may be also conducted on a periodic basis, resulting in a longer-term contract between Alice's services and *scribe*, in which case the immediate negotiation will be purely concerned with service parameters). *Scribe* may then negotiate on a much shorter-term basis with *es-trans* and other possible providers, with this negotiation being conducted on a 'per-document' basis.

Service delivery in this example is fairly clear-cut. It occurs when *es-trans* receives the original document and returns the translation. However, if this is not achieved within the specified timeframe (determined by the form in which Alice's service may have defined 'urgently' in this case), then either Alice's service or *scribe* may choose 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 and specialist forms of translation (such as a legal document) in order to seek a set of suitable translation services and open negotiation with them. A longer-term research goal is to be able to create entirely new services as oppose to predetermined ones. Our example does not contain an illustration of this form of composition, although we could easily extend it slightly to do so as follows. If Alice's need were to change, so that it was not only for translation of a legal document, but also for the registration of that document in some way, then there would be a need to extend the existing service rules to encompass such an option. (While this may not be an entirely new service, we envisage that many new services will consist of extensions and revisions to existing forms.)

Figure 3 - Web Services Stack Framework