Boxes: black, white, grey and glass box views of web-services

Steve Battle
Digital Media Systems Laboratory
HP Laboratories Bristol

HPL-2003-30
February 24th , 2003*

The purpose of this document is to contrast the different kinds of web service descriptions known as black, white, glass and grey-box views. In particular, the different assumptions underlying these views is claimed to be indicative of fundamental differences between DAML-Services (DAML-S), a DAML based Web Service Ontology, and the Web Service Modelling Framework (WSMF) that is central to the Semantic Web enabled Web Services (SWWS) project. Semantic web enabled web services are simply services with associated meta-data that helps us to understand what they do. By working with their semantics – the vocabulary is backed up by a model - the WSMF aims to support more robust web service interactions.

Boxes: black, white, grey and glass box views of web-services

Steve Battle, HP Labs, Bristol.[i]

The purpose of this document is to contrast the different kinds of web service descriptions known as black, white, glass and grey-box views. In particular, the different assumptions underlying these views is claimed to be indicative of fundamental differences between DAML-Services (DAML-S)[1], a DAML[2] based Web Service Ontology, and the Web Service Modelling Framework (WSMF)[3] that is central to the Semantic Web enabled Web Services (SWWS) project[4]. *Semantic web enabled web services* are simply services with associated meta-data that helps us to understand what they do. By working with their semantics – the vocabulary is backed up by a model – the WSMF aims to support more robust web service interactions.

Little boxes

Black boxes

Black-box approaches view a system purely in terms of observable inputs and outputs with the internals of the system hidden from view. One can make no assumptions about its behaviour or state beyond that specified by its interface. The black-box view deprives us not only of the ability to view the internal state, but also the rules by which the system is governed[5].

Black-box views can be extremely powerful. They are highly modular, so we get the desirable benefit that any system that supports an equivalent interface could be substituted for the original. The interface description confines itself to externally visible information. For example, we may describe messages that can be exchanged across the interface, and we can elaborate on these interactions by describing correlations between successive messages[6]. Additionally, we can attach pre- and post-conditions to these interactions, where these can again be expressed purely in terms of external entities. These interface descriptions are typically *declarative*, modelled in terms of the conditions that must hold at any time.

The black-box view falls down where we must discuss aspects of a service that do not appear in the interface. We cannot even discuss the state of the system. For example, an interaction with a service creates a pattern of permissions and obligations (e.g. The service provider has permission to deduct money from my account but then they are obliged to deliver the goods). It is arguable that it is impossible to really understand a service without reference to these concepts.

Black-box approaches also fall down where the system may interact with third parties in the course of its operation. Where the system presents multiple interfaces in this way, should we regard these other interactions as hidden from our view – are they, in fact, part of the system? But what if separately we have occasion to interact with these third-parties. This gives rise to the equivalent of the *three-body problem* of service interaction; sometimes we need to ascribe internal state to a system.

White Boxes

White boxes lie at the opposite extreme to black boxes, with the system internals being completely exposed to the user. However, this is an undesirable approach as we should be able to understand the service provided without being exposed to the full complexity of the implementation. While the white-box description defines exactly what the system does, it is over-specified. An ideal description should tell us only as

[i] Thanks also to my colleagues in Bristol who have unknowingly contributed to this paper.

much as is required to understand the system, but no more than is necessary; Abstraction is the key.

Grey Boxes

The Grey-box view stands in the middle ground between black- and white-box views. We move beyond the black-box view as soon as we have to introduce new entities to explain the behaviour of a process. A system is viewed as a grey-box if users have some knowledge of how it behaves behind the interface[7]. For example, this may include knowledge of a particular algorithm used in implementing the service.

With black-box testing we select test cases that fall inside the range of legal inputs/outputs as defined by its interface. This allows us to check that the system really conforms to that interface. By contrast, grey-box testing is carried out with some knowledge of the internals of the system (possibly gathered purely by experimentation), looking for edge-case limitations suggested by the actual implementation.

A more extreme characterization of the grey-box view, known as Grey-box integration[8], permits partial exposure of the system in terms of a coarse-grained operational model[9]. For example, in the context of Workflow Management Systems (WfMS), a given subsystem exposes its API to the WfMS and remains essentially a black box. This is set within an overall process description, used by the WfMS to enact the workflow. The resulting grey mixture is white on top, but black underneath.

While these examples appear difficult to reconcile at first glance, they do have a common theme. Where the white box view is fully exposed in all its gory details, the grey-box view allows only *partial exposure of the system behaviour*, whether by discovery or design.

Glass boxes

Both white- and grey-box views exist along a continuum that admits different levels of exposure of the system behaviour, defined in terms of its operational model. By contrast, the glass-box view attempts to retain the advantage of the purely *declarative* representation (expressed in terms of truth conditions) found in the black-box view. The aim is to provide support for meta-level reasoning about the process (i.e. rather than just executing it) by exposing the rules that govern it[10]. Put another way, a glass-box view exposes *what* the process does, without necessarily giving away *how* it does it. An analogy here is how we might interpret the recipe for a cake; we can simply follow the recipe and bake a cake, or we may examine the recipe to work out what ingredients we're missing.

The glass-box view really introduces a knowledge-level[11] approach to systems; a way to rationalise the behaviour of a system from the perspective of an external observer[12]. The system may be *introspective*, informing us about itself and what it is doing.

DAML Services

At the heart of the DAML-S ontology is the idea of the *Service.* We should begin by asking, what is a service? In the DAML-S conception of the world, services are modelled as single processes. They present a service profile that describes *what* they do, and a process model that describes in some way *how* they do it. They may also have a grounding that describes in concrete terms how the process is mapped onto a real web-service.

Among other things, a service is described in terms of its *ProcessModel*. DAML-S utilizes black boxes and glass boxes, as distinct ways of thinking about process modelling. It defines three distinct kinds of process: *AtomicProcess*; *SimpleProcess*; and *CompositeProcess*.

We explore the process model using an example taken from the DAML-S home-page; this is the Congo.com example of a fictitious online book-seller. We bring the example to life by providing graphical examples of resource description fragments that could be used.

Atomic Processes

Atomic processes are grounded in concrete web-accessible programs[13]. They are primitive and undecomposable, so in the absence of any prior knowledge about their implementation we can treat them only as black boxes.

The DAML-S process model is inherently a 2-party model[14]. Inputs & outputs are defined with respect to the service provider, so a service invocation provides input to the service and returns output to the user[ii]. The *LocateBook* process describes the initial interaction with the book-store where we might identify a desired book (by name) and obtain a description of it. Each call has a number of input parameters. For example, in Figure 1, *LocateBook* is described as an atomic process with an input, *bookName,* and an output, *bookDescription.* We can distinguish inputs from outputs by tracing the relevant sub-property relationships.

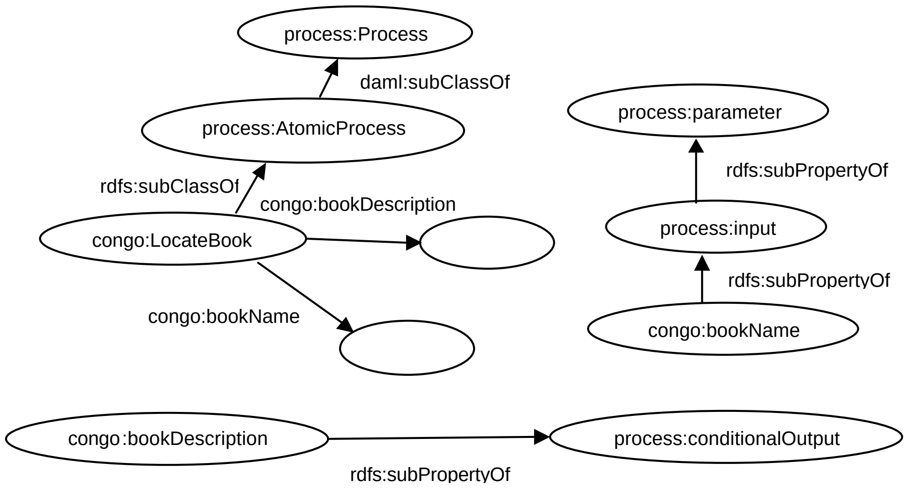

Figure 1: Atomic process

Simple Processes

By contrast with atomic processes, *simple* processes are not grounded and cannot be invoked. They are really abstract place-holders for more complex process compositions. For the sake of modelling, they may be conceived as executing in a single step[13]. For example, in addition to describing the concrete process of locating a book as described above, we would like to raise the discussion one level, to talk about the end-to-end process of book buying. In Figure 2 below, we create the *CongoBuy* process. It assumes a single input, the *congoBuyBookName* that would be relayed to *LocateBook*. The book description does not feature as an output, as it is only used internally to raise the appropriate order.

[ii] The relative ordering between input and output is not fixed by the process model. Different groundings permit different kinds of operational mapping (i.e. SOAP request-response, one-way, notification, and solicit-response).

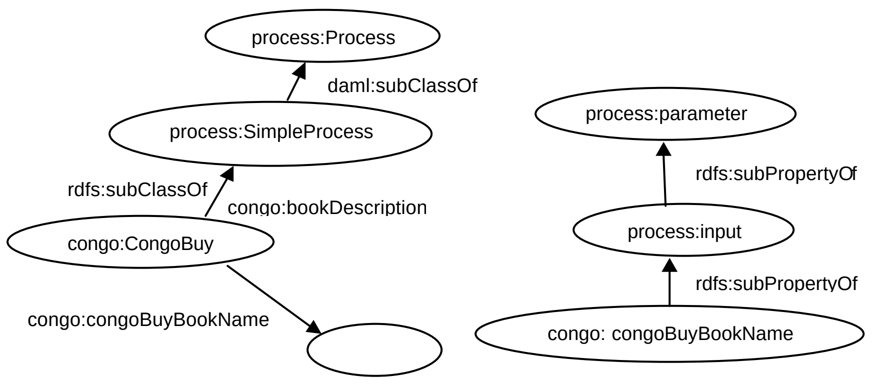

Figure 2: Simple process

As it stands, the simple model doesn't help us buy a book – it isn't a process we can directly invoke. We have to *expand* this simple process to describe the buying process in full.

Composite Processes

A DAML-S composite process defines a tree where each branch node includes a control construct, and each leaf node is a simple or atomic process. The example in Figure 3 shows an expanded process that first uses *LocateBook* as defined earlier, before proceeding to the checkout with *CongoBuyBook*. These things happen in a strict sequence, so this branch of the process tree uses a *Sequence* control construct. The components of the sequence are held in an ordered collection that can contain further sub-processes or control constructs (not shown).

The composite process can use control constructs including Sequence; Split providing concurrency (same as Concurrent or Parallel); Split-Join for subsequent resynchronization (same as Concurrent-Sync, Parallel-Sync, or Fork-join); Unordered (executing a bag of processes sequentially but in Any-Order); Choice (or Alternative) allowing a selection of processes; If-then-else for conditionals; and for iteration we may use Repeat-until, Repeat-while or Iterate (for infinite loops).

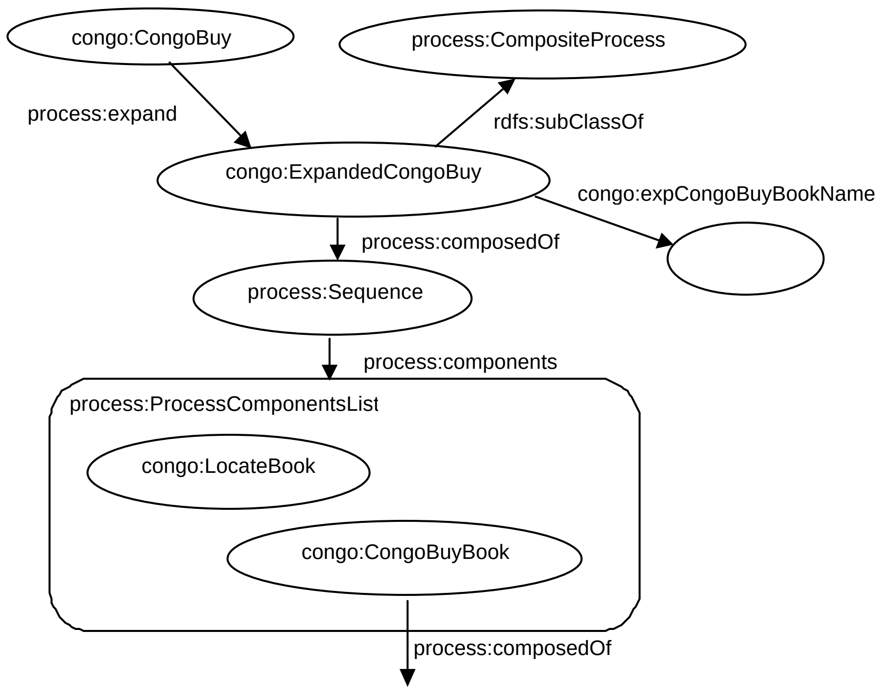

Figure 3: Composite Process

A DAML-S composite process is described as a glass-box view. If needs be, it can be viewed as a black box by *collapsing* it down to a simple process[15]. The aims of DAML-S are fundamentally about the creation of service ontologies for the purpose of sharing service descriptions in support of automated reasoning about them. For these reasons, the mapping into a declarative logical formalism is paramount.

The DAML-S composite process was not designed primarily as an operational model. It simply provides a vocabulary as a starting point for future work. If we want to execute a composite process, we have to go out of our way to add the required semantics[16,17]. Just as atomic processes must be grounded in concrete languages such as WSDL, we could devise a translation from a composite process into an executable process language such as BPEL4WS[18]. The strength of DAML-S and the glass-box view is that it supports reflection[19] about processes, rather than directly executing on them.

Web Service Modelling Framework

Control flow versus data flow

There is a darker side to DAML-S process modelling that limits the expressiveness of the language[20]. In addition to defining the control flow, the process model must define how data flows between processes. Even in the simple example shown, we have seen three different occurrences of the book-name parameter, one for each atomic, simple, and composite process that we have looked at. These input parameters must somehow be unified. Similarly, we may wish to unify the output of one sub-process with an input to the next, or to unify sub-process outputs with those of the composite.

The process:*sameValues* constraint is used to unify the parameters of *CongoBuy* and its expansion with the parameters of the concrete *LocateBook* process. This relation surfaced on the www-rdf-logic mailing list, "DAML-S Expressiveness Challenge #1" [21], and is documented in the new version of DAML-S 0.7[22]. The intention is that within an instance of a composite process, we unify the value of one property with another. Either of these properties may belong to the composite process itself, or to one of its sub-processes.

We should be clear that these inputs and outputs represent the abstract content of the message, not the concrete message itself[13]. With WSDL[23] groundings, all inputs to a process must be mapped onto a multi-part message; similarly for the outputs. There appears to be some 'wiggle room' that allows a parameter like the book name to be submitted to *CongoBuy* in one format but be passed on to *LocateBook* in another. The service description is independent of the syntax of any particular message. Therefore, there is room in DAML-S for data mediation, a key concept in the WSMF.

The WSMF breaks from the DAML-S model by taking a more data-centric approach. Where DAML-S associates process inputs and outputs more or less directly with messages in the service grounding, the WSMF will utilise semantic models of these data resources, making extensive use of ontologies to model the data. The unification of one data resource with another need not require syntactic equality at the level of sameValue, but can be achieved if there is a suitable ontological equivalence.

Where DAML-S primitives target the control flow, with data flow a secondary concern, the data-centric WSMF primitives emphasize data flow over control flow. The primitives it defines, such as split and join[3], describe the distribution and merging of content, rather than process concurrency. Concurrency is implicit in the flow graph model, and we can assume that all sub-processes execute concurrently. Indeed, if we don't want concurrency we have to re-introduce sequencing with explicit signalling. In Figure 4 we attempt to illustrate these flows between processes (however a simple diagram like this ultimately fails to capture the dynamic nature of the flow). Taking the Congo.com example, the *bookName* flows into *LocateBook* while the *bookDescription* flows out of it. To make the example more exciting, we throw in a desired quantity with the book description. The join implies more than the simple creation of a multi-part message that we find in the DAML-S/WSDL grounding[24]; it requires the construction of a single document integrating information from both inputs.

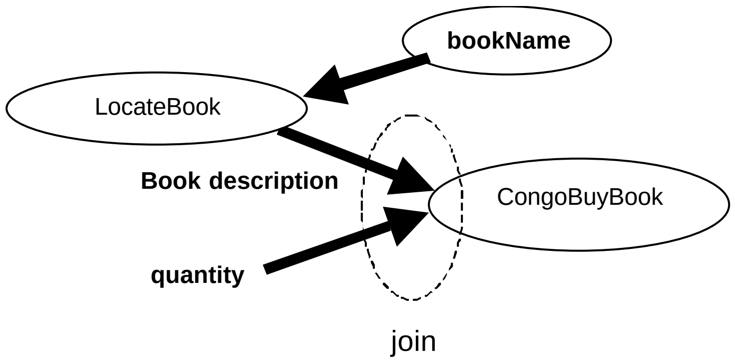

Figure 4: WSMF data flows

DAML-S and the WSMF come from opposite poles in their approaches to managing the flow of control versus the flow of data. It may be on this basis that we can foresee a future in which the two may be integrated.

abstract versus executable processes

The DAML-S/WSDL service grounding provides an account of how atomic processes are mapped onto web-service operations, associating DAML-S parameters and outputs with particular input/output ports on a web-service interface. These atomic processes correspond to the smallest steps of the service that are exposed to the outside world. As with the Congo.com example, these steps must somehow be linked to form what is in effect a business process; by this we mean an outward facing model of a process, described only in terms of external conversations[25]. DAML-S provides us with the composite process, but remains unclear as to how they should be interpreted. On the face of it, the composite process describes the *internal* structure of a complex service. The ambiguity arises because this does not inform us about who is responsible for the composition. If we return briefly to our cake recipe analogy, then the difference is between going to a shop to buy a ready-made cake (server-side composition), and doing it yourself (client-side composition). You could even employ an *agent* to make it for you, but what matters is where the responsibility resides. In all cases the outcome should be more or less equivalent – depending upon your skills as a baker. The conversation is the trail of atomic steps that are executed. However, whereas in the former interpretation the composite service is seen as a wrapper obscuring the details of the trail, in the latter interpretation the conversation is fully exposed to the client.

DAML-S does not differentiate between these two modes of use, which are described in BPEL4WS as executable and abstract processes. Executable processes model the actual behaviour of a participant (service), while abstract processes focus on the mutually visible message exchange behaviour, or business protocol[18]. The WSMF, if it is to focus on the service interface rather than on its internal representation, should address primarily these abstract processes.

Conclusion

The black-box view introduces an almost clinical isolation between the front and the back-end. This approach lends itself most naturally to interface led development, where the interface is developed first and will encompass a wide range of possible realizations. This situation would arise if we were trying to agree a standard interface in collaboration with partners. The black-box would represent the as yet unrealised service.

The grey-box view is more concerned with the pragmatics of the service implementation. It recognises the need of service providers to be able to code to abstract interface definitions without being absolutely conformant in every possible way. For example there may be real-time constraints on a given implementation that are more restrictive than the abstract interface would lead us to believe.

The greying of the interface provides a real problem for automation. With innumerable clients cut to the clean lines of the black-box, how can they cope gracefully with the variations introduced along the way as service providers adopt and adapt an interface.

The glass box view can serve the role of bridging between the two. It captures the semantics of the interface, providing a declarative way to represent the contingencies of the service instance. This *contingent* interface can be described as a specialisation of the abstract interface. We can use specialisation either to expand the range of behaviours beyond those described by the abstract interface, or to restrict the existing range of permitted behaviours (by adding constraints).

Our hypothesis is that reasoning, based on glass box views, can support interactions between partners who can agree what it is they want to achieve, without first knowing how. This approach should be resistant to minor syntactic variations in service interfaces, or even to more significant changes, at least where a common semantics can be established.

---

[1] DAML Services, http://www.daml.org/services
[2] The DARPA Agent Markup Language, http://www.daml.org
[3] Fensel and Bussler, The Web Service Modelling Framework, http://www.cs.vu.nl/~dieter/wese/wsmf.paper.pdf
[4] http://swws.semanticweb.org
[5] artificial intelligence laboratory, University of Michigan, Black Box Approach, http://ai.eecs.umich.edu/cogarch0/common/prop/blackbox.html
[6] W3C, Web Services Conversation Language, http://www.w3.org/TR/wscl10/
[7] A. Arpaci Dusseau and R. Arpaci Dusseau , Information and Control in Gray-Box Systems, http://www.cs.ucsd.edu/sosp01/papers/arpacidusseau.pdf
[8] D. Tombros, A. Geppert , Managing Heterogeneity in Commercially Available Workflow Management Systems: A Critical Evaluation , http://www.ifi.unizh.ch/dbtg/Projects/SWORDIES/PubDocs/Bericht4.ps
[9] M.Büchi, A Plea for Grey-Box Components, http://www.cs.iastate.edu/~leavens/FoCBS/buechi.html
[10] artificial intelligence laboratory, University of Michigan, Glass Box Approach, http://ai.eecs.umich.edu/cogarch0/common/prop/glassbox.html
[11] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87-127, 1982.
[12] The knowledge level according to Newell, http://arti4.vub.ac.be/memos/AI-Memo-93-09/section3.2.html
[13] DAML-S: Web Service Description for the Semantic Web, http://www.daml.org/services/ISWC2002-DAMLS.pdf
[14] DAML-S Design Rationale and Outstanding Issues, http://www.daml.org/services/daml-s/2001/10/rationale.html
[15] DAML-S: Semantic Markup for Web Services, http://www.daml.org/services/daml-s/2001/05/daml-s.html

[16] A.Ankolekar, F.Huch, K.Sycara, Concurrent Semantics for the Web Services Specification Language DAML-S, Coordination models and languages, 2002.
[17]Comparison of DAML-S and BPEL4WS, http://www.ksl.stanford.edu/projects/DAML/Webservices/DAMLS-BPEL.html
[18] Business Process Execution Language for Web Services, Version 1.0, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
[19] Review about Computational Reflection, http://tunes.org/Review/Reflection.html
[20] DAML-S Design Rationale and Outstanding Issues, http://www.daml.org/services/daml-s/2001/10/rationale.html
[21] www-rdf-logic archive, http://lists.w3.org/Archives/Public/www-rdf-logic/2001Jul/0029.html
[22] DAML-S 0.7 Draft Release, http://www.daml.org/services/daml-s/0.7/
[23] Web Services Description Language (WSDL) Version 1.2, http://www.w3.org/TR/wsdl12/
[24] Grounding.daml, http://www.daml.org/services/daml-s/0.7/Grounding.daml
[25] N.Apte and T.Mehta, Web Services, HP books Prentice Hall, 2002.