Pattern-Based Specification and Validation of Web Services Interaction Properties

Zheng Li, Jun Han, and Yan Jin

Faculty of ICT, Swinburne University of Technology,
John Street, Hawthorn, Melbourne, Victoria 3122, Australia
{zli, jhan, yjin}@ict.swin.edu.au

Abstract. There have been significant efforts in providing semantic descriptions for Web services, including the approach as exemplified by OWL-S. Part of the semantic description in OWL-S is about the interaction process of the service concerned, and adopts a procedural programming style. We argue that this style of description for service interactions is not natural to publishing service behavior properties from the viewpoint of facilitating third-party service composition and analysis. In this paper, we introduce a declarative approach that better supports the specification and use of service interaction properties in the service description and composition process. This approach uses patterns to describe the interaction behavior of a service as a set of constraints. As such, it supports the incremental description of a service's interaction behavior from the service developer's perspective, and the easy understanding and analysis of the interaction properties from the service user's perspective. We also introduce a framework and tool support for monitoring and checking the conformance of the service's run-time interactions against its specified interaction properties, to test whether the service is used properly and whether the service fulfils its behavioral obligations.

1 Introduction

Service-Oriented Computing (SOC) is emerging as an important paradigm for IT architectures and applications. A service provider publishes the interface description of the service in a registry, through which a user may search and access the description to locate the required services. The interface description of a Web service serves as the contract of interaction with its consumers and is the place where a consumer can find information about the service. In general, such a contract should cover issues beyond interface signatures, including functionality, quality and interaction behavior of the service. The more information about the service is provided, the more likely the service will be properly understood and utilized. However, the current Web service description standard - WSDL, only specifies the location and operation signatures of a service, but lacks the mechanisms for capturing its behavioral properties. This may cause significant problems regarding behavioral interoperability when the service is used. Without the behavioral properties or knowledge of a service, the consumer may make incorrect assumptions about the service, which may lead to interaction failure. As such, a rich service description model is needed to publish the observable behavior of Web services in general and its interaction protocols in particular, so that the consumer can have a better understanding of the service execution semantics and know how to interact with the service in a proper manner [12].

OWL-S is a prevailing rich description model for Web services. Its service model describes the interaction behavior of a service by viewing the service as a process. It provides a set of control constructs such as sequence, split, split+join etc. to specify the possible execution flow of a service's operations. The service model employs a procedural/imperative programming approach, and specifies step by step the process that the service will perform to reach a particular result. Although this procedural approach is suitable to certain situations, it has obvious limitations in characterizing services with diverse behaviors because the resulting process model will become too complex as part of the interface description. A complex interface description is difficult to comprehend, process and therefore use.

We argue that a rule-based declarative approach provides a better choice as it requires much simpler description, needing only one-third to one-sixth of the statements required by the procedural approach, when representing the same behavior [10]. In fact, a declarative style conveys the "what" rather than the "how" of the procedural style, and is consistent with the intention of service (interaction) description, i.e., "what" the service (interaction) behavior is. In addition, describing a service in a declarative manner enables the consumer to use the service in ways that the service designer does not foresee [11]. It also gives better support to automatic reasoning-based validation of the composition of multiple services with diverse behaviors [18]. For frequently changing services, adding and removing rules require much less effort than modifying existing procedural definitions. This is a very useful feature in the service design process, which always involves many iterations of modification and revision on the service behavior definition.

In this paper, we introduce a declarative approach to specifying the interaction behavior of Web services as interaction constraints. Each constraint states an occurrence or sequencing properties of a service's operation invocations, representing a partial view of the service protocol on the invocations. As such, this approach allows incremental specification of a service's interaction protocol.

This approach is based on our previous research on interaction constraint specification for software components [14, 15], which advocates the use of the property specification pattern system proposed by Dwyer *et al.* in [9] in order to give software practitioners easy access to the specification approach. As the basis for the interaction constraint specification for Web services, we develop an OWL-based ontology for the property patterns in this paper. We add this ontology to OWL-S as an enhancement and an alternative to its procedural style definition of service interaction behaviors.

We further introduce a framework and tool to monitor and check the conformance of a service's run-time behavior against the specified service interaction constraints. The framework employs finite state automata (FSA) to represent semantically an interaction constraint, utilizes a SOAP message monitor to track the run-time interactions of the service, and includes a validation module that checks the interactions against the FSAs (*i.e.* interaction constraints) for error detection.

The remainder of the paper is structured as follows. In section 2, we give a reference example as a basis for further discussion. Section 3 presents our constraint-based approach to specifying service behavioral properties together with the ontology for the interaction property patterns. Section 4 introduces the validation framework and its implementation. We then discuss the related work in section 5 before drawing some conclusions in section 6.

2 A Reference Example

Let us consider an auctioneer service that provides auction services on the Web. The auctioneer publishes its interface description in WSDL and communicates with a number of bidders and sellers by exchanging SOAP messages. The service is able to accept registrations from new bidders/sellers and hold auctions among registered bidders. It provides several operations to allow users to query the information of auction items, register and un-register themselves to the service, login and logout the service, bid or sell an item. The service also provides an operation allowing bidders to retract their previous bids. Figure 1 shows an excerpt of the WSDL description for the auctioneer.

```
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/Auctioneer"
  ……
  <wsdl:portType name="Auctioneer">
    <wsdl:operation name="opRegister" parameterOrder="userInfo">
       <wsdl:input message="impl:opRegisterRequest" name="opRegisterRequest"/>
       <wsdl:output message="impl:opRegisterResponse" name="opRegisterResponse"/>
    </wsdl:operation>

    <wsdl:operation name="opUnRegister" parameterOrder="userInfo">
       <wsdl:input message="impl:opUnRegisterRequest" name="opUnRegisterRequest"/>
       <wsdl:output message="impl:opUnRegisterResponse" name="opUnRegisterResponse"/>
    </wsdl:operation>

    <wsdl:operation name="opLogin" parameterOrder="userInfo">
       <wsdl:input message="impl:opLoginRequest" name="opLoginRequest"/>
       <wsdl:output message="impl:opLoginResponse" name="opLoginResponse"/>
    </wsdl:operation>

    <wsdl:operation name="opLogout" parameterOrder="userInfo">
       <wsdl:input message="impl:opLogoutRequest" name="opLogoutRequest"/>
       <wsdl:output message="impl:opLogoutResponse" name="opLogoutResponse"/>
    </wsdl:operation>

    <wsdl:operation name="opBid" parameterOrder="userInfo itemNo price">
       <wsdl:input message="impl:opBidRequest" name="opBidRequest"/>
       <wsdl:output message="impl:opBidResponse" name="opBidResponse"/>
    </wsdl:operation>

    <wsdl:operation name="opRetract" parameterOrder="userInfo bidRefNo">
       <wsdl:input message="impl:opRetractRequest" name="opRetractRequest"/>
       <wsdl:output message="impl:opRetractResponse" name="opRetractResponse"/>
    </wsdl:operation>

    <wsdl:operation name="opSell" parameterOrder="userInfo itemInfo">
       <wsdl:input message="impl:opSellRequest" name="opSellRequest"/>
       <wsdl:output message="impl:opSellResponse" name="opSellResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  ……
</wsdl:definitions>
```

Fig. 1. Excerpt of the Auctioneer Web Service Description in WSDL 1.1

3 Pattern-Based Interaction Property Specification

In this section, we first introduce our pattern-based approach to specifying the interaction behavior of Web services. We then define an ontology for the pattern system to provide the semantic basis for such service behavior description. An example is given to illustrate how the ontology is used to define service interaction constraints.

3.1 Property Specification Patterns

Our approach to defining interaction constraints for Web services builds on the property Specification Pattern System (SPS) proposed by Dywer *et al.* in [9]. The SPS patterns were originally developed as "high-level specification abstractions" to assist practitioners to formally specify system properties. The authors showed in [9] that SPS is able to cater for a majority of system properties.

In our approach, the SPS patterns and scopes are used to define basic and higher-level operators used to specify the occurrence and sequencing rules about invocations to a Web service. The introduction of SPS is aimed to facilitate the use of formal methods by Web service developers in describing the service interaction constraints or protocol. Precisely defined constraints are essential to ensure the proper use of the services when composing business applications or processes. In Figure 2, we list the basic pattern and scope operators used in our work as well as their usage, where op_1, …, and op_4 are distinct operations and n is a natural.

$$\left.\begin{array}{l} op_1 \text{ is absent} \\ op_1 \text{ exists [} \begin{bmatrix}\text{at most}\\ \text{at least}\end{bmatrix} n \text{ times]} \\ op_1 \text{ precedes } op_2 \\ op_1 \text{ leads to } op_2 \end{array}\right\} \times \left\{\begin{array}{l} \text{globally} \\ \text{before } op_3 \\ \text{after } op_3 \\ \text{after } op_3 \text{ until } op_4 \\ \text{between } op_3 \text{ and } op_4 \end{array}\right.$$

Fig. 2. Pattern and Scope Operators

Specifically, for a Web service, we provide SPS patterns for specifying the restrictions on both the occurrence of individual operations' invocations and the order (or sequencing) between different operation invocations. The occurrence patterns include *absence*, *existence*, and *bounded existence*. In particular, the absence pattern requires that invocations to the given operation not occur (within the given scope). The existence pattern states that invocations to the given operation must appear. The bounded existence pattern extends it with lower and upper bounds on the number of invocations. For example, to control the overall system performance, the auction service provider may want to set a limit on the number of bids that a bidder can make during each session. This can be stated as:

opBid **exists at most 3 times after** *opLogin* **until** *opLogout;*

where the upper bound is 3 (see below for explanations of the "after-until" scope).

The sequencing patterns include *precedence*, *response*, *precedence chain*, and *response chain*. For instance, a precedence property of the auctioneer service "*opRegister* **precedes** *opUnRegister*" states that there must be at least one *opRegister* invocation before any *opUnRegister* invocation. One may think *opRegister* enables *opUnRegister*. A response property "*opLogin* **leads to** *opLogout*" states that an *opLogin* invocation must eventually be followed by an *opLogout* invocation. In essence, this specifies a cause-effect relationship between *opLogin* and *opLogout*.

To handle more complex properties, SPS patterns can be associated with various scopes such as *global*, *before*, *after*, *between-and*, and *after-until*. Each scope specifies a portion(s) of a service's interaction history, in which the given pattern takes effect. More specifically, the *global* scope refers to the entire history. The *before* scope refers to the initial portion of the history up to the first occurrence of an call message of the given operation. The *after* scope however states the inverse, *i.e.* the portion after the first occurrence of a reply message of the given operation. In the *between-and* scope, each portion is marked between consecutive occurrences of two messages. The starting message is the reply message of the first given operation, while the ending message is the call message of the second given operation. The *after-until* scope is similar but allows the portion to be open ended. That is, the given pattern continues to take effect after a reply to the first operation, even if the second operation will never be invoked afterwards. In contrast, in the *between-and* scope, the second operation has to be invoked in order for the given pattern to be applicable.

In the above, we have assumed operations be the atomic unit of concern. To cope with realistic services, however, one needs to consider the effect of different parameter values on the service interaction logic. Therefore, we allow conditions to be associated with each constraint specification to fine-tune the specified relationship. For example, the earlier constraint on the upper bound of bids by *each bidder* can be elaborated as:

opBid **exists at most 3 times after** *opLogin* **until** *opLogout*
    where *opBid.userInfo = opLogin.userInfo = opLogout.userInfo;*

As detailed later, we make use of the Semantic Web Rule Language (SWRL) to state such conditions.

Note that, in general, SPS patterns can be nested to describe complex constraints [9]. For simplicity, we do not explicitly deal with pattern nesting in this paper. It is however easy to accommodate them in our specification approach.

The following are two further example constraints for the auctioneer service:

opBid **precedes** *opRetract* **after** *opLogin* **until** *opLogout*
    where *opBid.userInfo = opRetract.userInfo = opLogin.userInfo = opLogout.userInfo*
    and *opBid.bidRefNo = opRetract.bidRefNo;*

opSell **precedes** *opBid* **where** *opSell.itemNo = opBid.itemNo;*

The first constraint states that if a bidder is to retract a valid bid (*opRetract*), there must be a preceding successful bid (*opBid*) by the same bidder in the same session. The second constraint says that a bidder can only bid for items on sale.

3.2 An Ontology for Interaction Property Patterns

The patterns and scopes used to specify service interaction constraints are defined in the ontology for Interaction Property Patterns (IPPs). It provides a common terminology for service developers to specify the interaction constraints of Web services in a standard and formal way.

The IPP ontology is defined using OWL and is designed as an add-on to OWL-S as a complement to the Service Model. More specifically, the topmost class defined in this ontology, *InteractionContract*, serves as an alternative to OWL-S *CompositeProcess* class. Figure 3 depicts the relationship between the IPP ontology and OWL-S. As shown, *InteractionContract* is embedded in the Service Model and uses the *AtomicProcess* class as the basic entities to define the interaction constraints of Web services in a rule-based/declarative manner. Note that the Service Profile and Service Grounding are not affected.

Figure 4 presents all the classes and their relationships as defined in the IPP ontology, where classes are drawn as ovals and properties are depicted as arc labels. Note that the shaded classes are not part of the IPP ontology, but are defined in OWL-S or XML Schema.

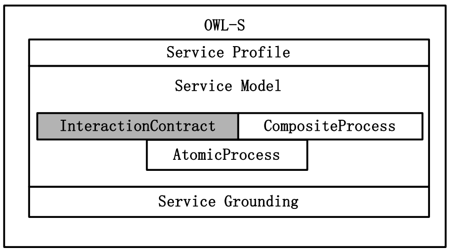

Fig. 3. Relationship between the IPP ontology and OWL-S

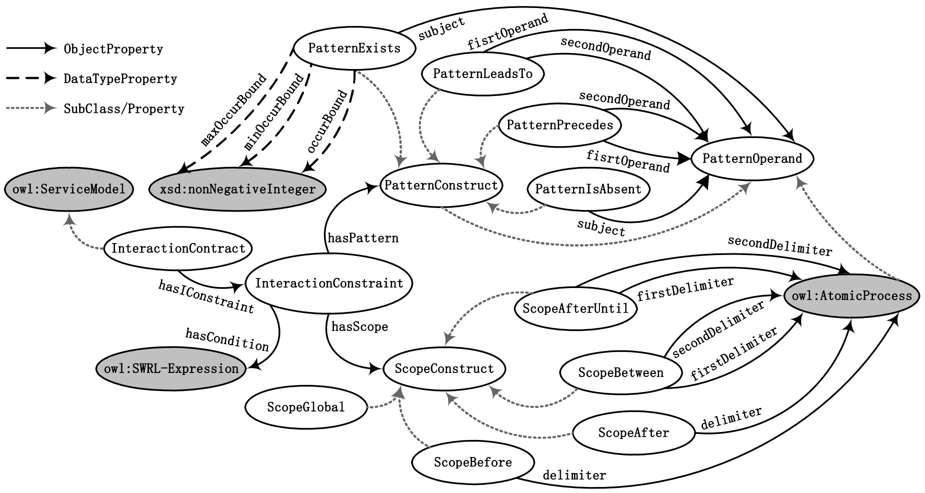

Fig. 4. Classes in the IPP Ontology

Class *InteractionContract.* In the IPP ontology, *InteractionContract* is the topmost concept denoting all the interaction constraints for a service. *InteractionContract* is defined as a subclass of OWL-S *ServiceModel.* It has a *hasIConstraint* property, specifying an *InteractionConstraint* instance.

Class *InteractionConstraint.* The *InteractionConstraint* class has three properties, *hasPattern, hasScope* and *hasCondition.* The *hasPattern* property ranges over the class *PatternConstruct.* Its value specifies an occurrence or sequencing rule over some operations' invocations. The *hasScope* property ranges over the class *ScopeConstruct.* Its value indicates the scope over which the specified rule applies. The *hasCondition* property specifies an *SWRL-Expression* (defined in the OWL-S) for the condition governing operation parameter values.

Class *PatternConstruct.* *PatternConstruct* is the superclass of four pattern classes: *PatternIsAbsent*, *PatternExists*, *PatternPrecedes* and *PatternLeadsTo.* Each of them is used to express one specific type of the service behavior. *PatternIsAbsent* has one property *subject* that names the operation of concern. The value of *subject* is an instance of *PatternOperand* that can be of either type OWL-S *AtomicProcess*, or *PatternConstruct.* The latter enables potential pattern nesting. The *subject* property of *PatternExists* is similar. In addition, *PatternExists* has three cardinality properties: *maxOccurBound*, *minOccurBound* and *occurBound* used to restrict the number of invocations to the operation of interest. All these properties are of the XML schema data type: *xsd:nonNegativeInteger.* Well-formedness rules about their occurrences are straightforward and thus omitted here. Both *PatternPrecedes* and *PatternLeadsTo* have two properties, *firstOperand* and *secondOperand,* ranging over *PatternOperand.*

Class *ScopeConstruct.* As noted earlier, the *ScopeConstruct* class is used to indicate a portion of the interaction history over which the constraint must be satisfied. There are five *ScopeConstruct* subclasses: *ScopeGlobal*, *ScopeBefore*, *ScopeAfter*, *ScopeBetween*, *ScopeAfterUntil.* Each of these classes defines zero, one or two delimiters, specifying the starting and ending operation invocations or replies. It is worth noting that the scope within which the constraint is evaluated starts, if applicable, after the reply message of the first operation is received, and finishes, if applicable, before the call message of the second operation is received.

Class*SWRL-Expression.* As noted above, we use the *SWRL-Expression* class to specify conditions for interaction constraints. The detail of this class can be found in OWL-S and is thus not repeated here.

3.3 Example

To illustrate the use of the IPP ontology, consider the auctioneer Web service. As discussed earlier, assume that a user can only bid at most 3 times within each of his logins. This means the *opBid* operation can only be invoked at most 3 times after the user successfully invokes *opLogin* and before he invokes *opLogOut.* Figure 5 shows the definition of this constraint according to the IPP ontology.

```
<ipp:InteractionConstraint>
    <ipp:hasPattern>
        <ipp:PatternExists>
          <ipp:subject>
              <process:process rdf:resource="#opBid"/>
          </ipp:subject>
          <ipp:maxOccurBound rdf:datatype="&xsd;#nonNegativeInteger">3</ipp:maxOccurBound>
        </ipp:PatternExists>
    </ipp:hasPattern>

    <ipp:hasScope>
        <ipp:ScopeAfterUntil>
          <ipp:firstDelimiter>
              <process:process rdf:resource="#opLogin"/>
          </ipp:firstDelimiter>
          <ipp:secondDelimiter>
              <process:process rdf:resource="#opLogOut"/>
          </ipp:secondDelimiter>
        </ipp:ScopeAfterUntil>
    </ipp:hasScope>

    <ipp:hasCondition>
        <expr:SWRL-Expression>
           <expr:expressionBody rdf:parseType="Literal">
            <swrl:AtomList>
              <rdf:first>
                <swrl:sameIndividualAtom>
                  <swrl:argument1 rdf:resource="#opBidUserInfo"/>
                  <swrl:argument2 rdf:resource="#opLoginUserInfo"/>
                  <swrl:argument3 rdf:resource="#opLogOutUserInfo"/>
                </swrl:sameIndividualAtom>
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;#nil"/>
            </swrl:AtomList>
          </expr:expressionBody>
        </expr:SWRL-Expression>
    </ipp:hasCondition>
</ipp:InteractionConstraint>
```

Fig. 5. An Example Interaction Constraint Definition

4 Runtime Validation of Interaction Constraints

Explicit specification of Web service interaction constraints helps the service designer and the service client to implement and use a service properly. Whether or not a service is actually used correctly at run-time is a different question. Validation or testing is often required. In this section, we introduce a framework and a tool that allows us to validate the interactions with a Web service at run-time against its pre-defined interaction constraints.

4.1 Validation Framework

Our validation framework and tool monitor and validate the messages received and sent by a service against its interaction constraint specifications. Its message monitoring and interception builds on Web service platforms and tools. Its validation mainly makes use of the tool implementation of [14]. The monitoring and validation process is fully automated at run time. Figure 6 shows the overall validation architecture. The key techniques used include:

– Translating the constraint specifications into finite state automata (FSAs) that serve as the constraints' internal representation for easy processing;
– Identifying and intercepting the run-time messages exchanged with a Web service;

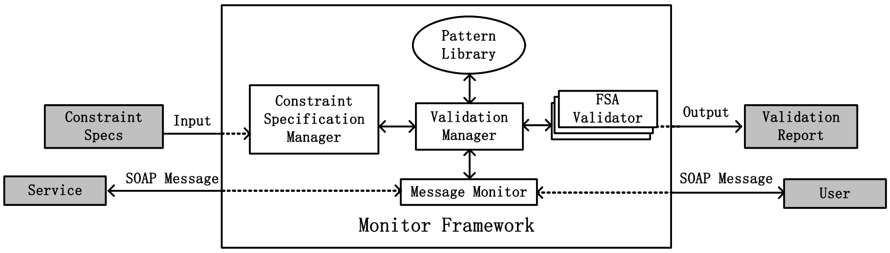

Fig. 6. Validation Framework

– Advancing the effective constraint FSAs using the intercepted message, and reporting violations, if any.

The monitoring framework consists of five components: Validation Manager (VM), Constraint Specification Manager (CSM), FSA Validators (FVs), Message Monitor (MM) and Pattern Library (PL), with VM coordinating all the other four components. PL maintains all the patterns and scopes and their FSA semantics. CSM reads the interaction constraint specifications embedded in the OWL-S service description file, translates them into an internal format. MM observes the incoming and outgoing SOAP messages of the Web service and intercepts the run-time operation invocations. All the SOAP messages exchanged between the service and the user are logged and forwarded to VM. Upon receiving a message, VM queries CSM to get all relevant constraint specifications. If the corresponding FVs have not been created, VM initialize them based on the used patterns and their FSA semantics stored in PL. It then asks all the relevant FVs to check the intercepted operation invocation message against their internal FSAs. If the message is not acceptable to any FSA, a violation report is issued.

4.2 Constraint Representation

The semantics of interaction constraints is informally given in section 3. To enable tool support, their semantics needs to be precisely defined. To do so, we choose FSAs as their formal semantic representations. When involving no condition about parameter values, in general, each interaction constraint has a corresponding FSA representation where arc labels are sets of operation call or reply messages. Such a FSA can be constructed prior to the first relevant message being identified. When a “where” condition is stated, an interaction constraint corresponds to a number of FSAs, each for a possible value combination of the parameters. Such an FSA is dynamically instantiated only when a parameter value of interest is observed. Further details about the FSA representation can be found in [13, 14]. We illustrate below the FSA representation of constraints using the earlier example on the bounded existence of bids (Figure 5).

Figure 7 shows the FSA corresponding to this example constraint, where opBid_{b1} denotes the set of all *opBid* call and reply messages exchanged with bidder b_1 (i.e., *opBid.userInfo* refers to b_1 as the ID). $\mathrm{opLogin}_{b1}^{reply}$ is the set of *opLogin* reply messages

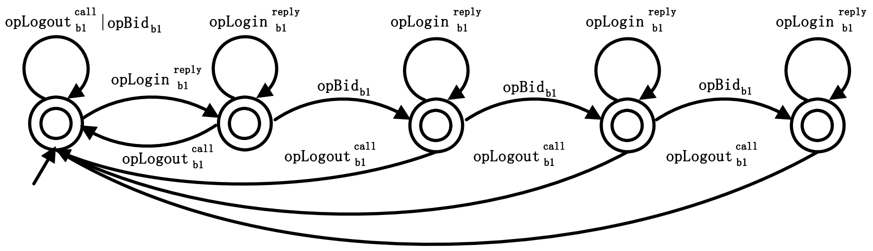

Fig. 7. FSA for the Example Interaction Constraint of Figure 5

to b_1. $opLogout^{call}_{b1}$ is the set of *opLogout* call messages from b1. Note that we have omitted all the other messages that can be received at every state for brevity. As shown, *opBid* cannot be invoked when the FSA enters the rightmost state until b_1 logs out and re-logs in.

4.3 Validation Process

The validation process starts when the Message Monitor detects a SOAP request/response message and forwards the message to the Validation Manager. Then VM finds out from CSM all the interaction constraint specifications in which such a message is of interest, and creates a FSA validator for each constraint using the message's parameter values, if such a FV does not already exist. VM then tries to advance the state of each of these relevant FVs using the observed message. An error or violation will be reported if the intercepted message is inhibited at the current state of any FSA. That is, the message does not appear in any labeling set of any outgoing arc of the current state. For example, an *opBid* call message received at the rightmost state of Figure 7 represents a constraint violation. If there is no interaction constraint in which an observed message is of interest, the message will be ignored by the Validation Manager.

4.4 Implementation

Our implementation of the run-time monitoring framework is based on open source platforms and tools. The reason behind this decision is that the source code is available and new features can be added if required. For our implementation, Tomcat 5 and Apache Axis 1.2 are used to set up a web server to run Web services. Tomcat is a lightweight HTTP server with all the features we need to run Web services. Axis provides an implementation of the W3C SOAP standard. They constitute a reliable and stable platform on which to implement Java Web Services.

In implementing the validation framework, we have reused the architecture of a runtime validation tool developed in [14] for CORBA-based systems, including the Pattern Library, Validation Manager and FSA Validator. However, these modules have been enhanced to better deal with the full range of interaction property patterns. We have also modified the Constraint Specification Manager module for processing the XML-based specifications of service interaction constraints. A new addition in this work is the Message Monitor that captures the SOAP messages (calls and returns) exchanged between a service and its user(s), and analyzes them at run-time as to their types, corresponding operations, etc. Part of it is a tool in the Axis package called SOAP Monitor, providing a way to intercept the SOAP messages. The SOAP Monitor utility adds one new handler to the global handler chain in the Axis architecture. As SOAP requests and responses go in and out of the service, the SOAP messages are forwarded to the SOAP Monitor service where it can be displayed using a web browser interface.

It is worth noting that our validation framework is not centralized. The Message Monitor (MM) resides on each server hosting services. The other parts of the framework can be deployed on the server, with the client or elsewhere. As long as the MM on the server side is working, one or more validation applications can be connected to MM, which enables multiple parties, such as service owner and users, to monitor and validate service behavior simultaneously.

5 Related Work and Discussion

Some proposed Web service standards, such as BPEL [3] and WSCDL [16], are composition languages in nature and specify service behavior from the service composition or business process point of view [6].What they specify is the required behavior for services rather than the behavior services actually provide.

Some ongoing research efforts recognize the needs for describing the behavior properties of individual services, but use rather abstract notations that are not suitable for service developers or users. [6-8] use a single finite state machine (FSM) to describe the overall observable behavior of a service. [8] focuses on protocol compatibility checking and [6, 7] extend FSMs by associating more properties to transitions. Such a FSM-based approach is good at describing services with simple behavior. However, when dealing with services with diverse behavior, this approach does not scale well with the increase in the number of states and transitions. The resultant FSMs can become difficult to understand and process. In contrast, our divide-and-conquer specification approach scales well with the number of constraints. On the other hand, [6, 7] deal with time-based service protocols. This can be potentially integrated with our work emphasizing inter-message relationships, resulting in more comprehensive service protocol descriptions.

[4, 19, 20] employ an ontological approach to specify interaction protocols. [4, 20] define ontologies for FSMs. Like [6-8], they use a single FSM to model each service behavior. Therefore, their approaches are subject to the same scalability limitation. Whereas in our approach, the FSA is only used for run-time validation and we use interaction property patterns for service behavior specification. Furthermore, we use multiple constraints/FSAs to cover the full behavior of services, which offers modularity and better scalability. [19] uses ontologies to represent service operation, input, conditional/unconditional output, precondition, and conditional/unconditional effect as the behavior constraint of a service. This approach is not capable of expressing temporal sequencing interaction constraints.

A body of work on Web service monitoring has been reported. [17] proposes an approach to specifying and monitoring Service Level Agreements. It focuses on Quality of Service, and monitors such properties as performance and costs instead of interaction behavior. [5] aims to monitor service compositions at run-time to see whether services satisfy the assertions specified in the service composition defined by BPEL. The assertions are the requirements from the service consumer, rather than services' properties. In contrast, our monitoring framework intends to assess whether a service's behavior conforms to its designer's intent. In addition, our monitor attaches to the service itself rather than to a service consumer such as the BPEL process.

Also related to our approach is the work based on patterns. [1] provides a rich set to patterns that can be used to model workflow. The workflow patterns follow the procedural approach to interaction specification and are very similar to the ControlConstructs defined in OWL-S's Service Model. The approach we propose is declarative in nature and is aimed at addressing the limitations of procedural approach employed by OWL-S. The "Service Interaction Patterns" in [2] describes how an individual message or a request/response message pair is transferred between two or more parties, whereas our patterns describe the sequential order in which multiple messages or operation invocations may occur. They mainly look at message exchanges from a system point of view, while we primarily study message exchanges from an individual service's point of view. As such, these two approaches have different focuses.

When putting our approach into practical use, the service designer needs to ensure the consistency of all the interaction constraints of a service. Inconsistency among constraints will leads to a situation where calls to an operation will always violate some rules. This issue is discussed in [13] where consistency checking is done by testing the non-emptiness of the language intersection of the interaction constraints and proving that each operation has its role in the intersection.

6 Conclusion

In this paper, we have introduced a declarative constraint-based approach to specifying the observable behavioral properties of Web services. The approach employs intuitive patterns to help practitioners describe the interaction constraints of a Web service. The constraints conjunctively determine the behavioral properties of the service. We have defined an ontology for these patterns and embed it into the OWL-S framework, enabling pattern-based interaction behavior description for Web services.

We have also presented a framework that supports the monitoring and validation of the runtime interactions with Web services against their specified interaction constraints. This provides a useful tool for adjudicating whether a service's behavior conforms to its design and whether the service is being used properly. The tool is able to identify and report any violations of such nature.

Our future work will include considering required operations of services and static checking of interaction compatibility between services or between individual services and the service composition specification.

Acknowledgement. This work is partially supported by the Department of Education, Science and Training (DEST) grant (AU-DEST-CG060081) from the Innovation Access Programme - International Science and Technology established under the Australian Government's innovation statement, Backing Australia's Ability.

References

1. Workflow Patterns. www.workflowpatterns.com (2005)
2. Alistair Barros, M.D., Arthur ter Hofstede: Service Interaction Patterns. In Proc. 3rd International Conference on Business Process Management (2005) 302-318, Eindhoven, The Netherlands
3. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ (2003)
4. Ashri, R., Denker, G., Marvin, D., Surridge, M., Payne, T.R.: Semantic Web Service Interaction Protocols: An Ontological Approach. In Proc. Third International Semantic Web Conference, Vol. 3298 (2004) 304-319, Hiroshima, Japan
5. Baresi, L., Ghezzi, C., Guinea, S.: Smart Monitors for Composed Services. In Proc. International Conference on Service-Oriented Computing (2004) 193-202, New York City, NY, USA
6. Benatallah, B., Casati, F., Skogsrud, H., Toumani, F.: Abstracting and Enforcing Web Service Protocols. International Journal of Cooperative Information Systems Vol. 13 (4) (2004) 413-440
7. Benatallah, B., Casati, F., Toumani, F., Hamadi, R.: Conceptual Modeling of Web Service Conversations. In Proc. Advanced Information Systems Engineering (CAiSE), Vol. 2681 (2003) 449-467, Klagenfurt/Velden, Austria
8. Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic Composition of e-Services that Export their Behavior. In Proc. International Conference on Service-Oriented Computing (2003) 43-58, Trento, Italy
9. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in Property Specifications for Finite-state Verification. In Proc. International Conference on Software Engineering (1999) 411-420, Los Angeles, CA, USA
10. Gottesdiener, E.: Procedural versus declarative. Application Development Trends Magazine (1997)
11. Guillaume, D., Plante, R.: Declarative Metadata Processing with XML and Java. In Proc. Astronomical Society of the Pacific Conference Series, Vol. 238 (2001)
12. Han, J.: Interaction Compatibility: An Essential Ingredient for Service Composition. In Proc. International Workshop on Grid and Cooperative Computing (2003) 59-66, Shanghai, China
13. Jin, Y., Han, J.: Consistency and Interoperability Checking for Component Interaction Rules. In Proc. Twelfth Asia-Pacific Software Engineering Conference (2005), Taipei, Taiwan
14. Jin, Y., Han, J.: Runtime Validation of Behavioural Contracts for Component Software. In Proc. Fifth International Conference On Quality Software (2005) 177-184, Melbourne, Australia
15. Jin, Y., Han, J.: Specifying Interaction Constraints of Software Components for Better Understandability and Interoperability. In Proc. International Conference on COTS-Based Software Systems, Vol. 3412 (2005) 54-64, Orlando, Florida, USA
16. Kavantzas, N., Burdett, D., Ritzinger, G.: Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/ (2004)
17. Keller, A., Ludwig, H.: Defining and Monitoring Service-Level Agreements for Dynamic e-Business. In Proc. Conference on Systems Administration (2002) 189-204, Philadelphia, PA, USA

18. Lara, R., Lausen, H., Arroyo, S., Bruijn, J.d., Fensel, D.: Semantic web services: description requirements and current technologies. In Proc. International Workshop on Electronic Commerce, Agents, and Semantic Web Services, In conjunction with the Fifth International Conference on Electronic Commerce (ICEC) (2003), Pittsburgh, PA, USA
19. Sriharee, N., Senivongse, T.: Discovering Web Services Using Behavioural Constraints and Ontology. In Proc. International Conference on Distributed Applications and Interoperable Systems, Vol. 2893 (2003) 248-259, Paris, France
20. Toivonen, S., Helin, H.: Representing Interaction Protocols in DAML. In Proc. International Symposium on Agent Mediated Knowledge Management, Vol. 2926 (2003) 310-321, Stanford, CA, USA