Checking Complex Compositions of Web Services Against Policy Constraints

Andrew Dingwall-Smith and Anthony Finkelstein

London Software Systems, University College London
Malet Place, London, WC1E 6BT, UK

Abstract. Research in web services has allowed reusable, distributed, loosely coupled components which can easily be composed to build systems or to produce more complex services. Composition of these components is generally done in an ad-hoc manner. As compositions of services become more widely used and, inevitably, more complex, there is a need to ensure that compositions of services obey constraints. In this paper, we consider the need to provide policy constraints on service compositions, that define how services can be composed in a particular business setting. We describe compositions using WS-CDL and we use `xlinkit` to express policy constraints as consistency rules over XML documents.

1 Introduction

Policy constraints are constraints on a system which originate from the business needs of an organisation. Some examples might be that it is necessary to perform a credit check before certain other interactions can be performed or that certain parties in an interaction should not communicate with each other. Web services are extremely valuable in providing distribution and loose coupling and allowing application integration within an organisation but there is a need to ensure that compositions of these services are consistent with policy constraints.

WS-CDL[1] is an XML language for describing complex interactions and compositions of services. It is intended for modelling these interactions from a global perspective, independent of underlying implementations, rather than as an executable composition language such as BPEL. WS-CDL is currently a W3C candidate recommendation.

The problem which we tackle is the need to ensure that complex compositions of web services, described in WS-CDL, conform to service policy constraints. We are able to tackle this problem by using `xlinkit`, which checks consistency rules over XML documents, by expressing policy constraints in the `xlinkit` rule language.

We envision our work being used in the following way. Organisations will have a number of existing services which they regularly use and which they want to be able to integrate in various complex ways. They will be able to describe policy constraints, using our approach, which constrain how these services can be composed. Later, service composers will design compositions using WS-CDL and will be able to check, using `xlinkit`, that their service compositions obey the policy constraints on the services which they use.

The contribution of this work is a simple, lightweight approach, using existing technology, to check that descriptions of web service compositions, described in WS-CDL, conform to policy constraints. The approach has been successfully implemented using `xlinkit`.

In section 2 of this paper, we briefly introduce `xlinkit` and the rule language which it uses. In section 3 we describe our approach to checking policy constraints. In section 4 we describe how we check policy constraints relating to the types of interactions allowed between the roles in a choreography and in section 5 we describe our approach to constraints on the ordering of interactions. We review related work in section 6 and our conclusions and ideas for future work are presented in section 7.

2 Overview of **`xlinkit`**

To test consistency, we make use of `xlinkit`[2–4], a tool for checking consistency within and between XML documents. The CLiX[5] language is an XML language used to specify consistency rules for use with `xlinkit`. This language is based on first order logic and makes use of XPaths[6] to select sets of nodes within documents. The `xlinkit` tool includes a development environment to allow easier editing and development of rules. This tool allows graphical editing of rule structures and generates the CLiX rules.

An example of the CLiX language is shown in figure 1. This example is intended to apply to an XHTML document and requires that all links within the document link to the `www.example.org` domain. The `forall` element selects the set of all link elements (`a` elements which have an `href` attribute) in the document and uses the variable `link` to refer to these elements. The link elements are selected using the XPath in the `in` attribute'. The `equal` element requires that for each element referred to by the `link` variable, the two operands are equal. These two operands are also XPaths. The first operand selects the portion `href` attribute which should refer to the `www.example.org` domain. A reference to the `link` variable is made in this XPath by preceding the name of the variable by the `$` sign. The rest of the XPath is then relative to the element referenced by this variable. The second operand is a literal string value. These two values should be equal so that the start of the `href` attribute is equal to the second operand (i.e it should point to the `www.example.org` domain).

```
<forall var="link" in="//a[@href]">
  <equal op1="substring($link/@href, 1, 22)"
         op2="'http://www.example.org'"/>
</forall>
```

Fig. 1. A CLiX rule which requires that all links point to the 'www.example.org' domain name.

CLiX rules can make use of the universal quantifier (forall) and existential quantifier (exists). These quantifiers have an XPath, which is used to select a set of nodes, and a variable name. When the rule is evaluated against a set of documents, the path is evaluated, resulting in a set of XML nodes which satisfy the XPath. The node set is then iterated over and the quantifier variable is bound to each node in the node set in turn and the rule within the quantifier element is evaluated for each value of the variable.

The CLiX language also makes use of seven standard operators (equal, not equal, greater, less, greater or equal, less or equal and same). All of these are standard mathematical operators, except for the 'same' operator, which evaluates to true if two XPaths refer to the same document node (all other operators work on the value of the nodes). The CLiX language supports the inclusion of custom operators in addition to the standard operators. In `xlinkit`, the operators are defined using ECMAScript[7]. Operators are combined to build more complex rules using and, or, not, implies and "if and only if" operators.

The `xlinkit` rule checking engine checks constraints for single documents or sets of documents. If a set of documents is provided as input then XPaths are evaluated for every document. This allows consistency rules between documents to be checked. If violations are detected, a report is generated which identifies which elements in the documents violated the constraints.

3 Checking Policy Constraints

The behaviour of web service compositions is implemented using orchestration languages, such as BPEL, and general purpose languages, such as Java and C#. A web service composition may be composed of heterogeneous services which are able to interoperate through web service standards. It is difficult to check consistency between policy constraints and these languages as many different languages can be used. Some implementations are also likely to be too low level to easily allow consistency checking to take place, particularly where general purpose languages are used. Furthermore, it is only possible to obtain a local view by examining web service orchestrations, whether implemented in an orchestration language or a general purpose language. By looking at an orchestration it can be seen how a web service interacts with the partners it provides services to and those it consumes services from. It is not possible to examine more complex interactions involving multiple levels of composition.

Using WS-CDL, it is possible to model a web service composition from a global perspective. In [1], WS-CDL is described as follows:

> The Web Services Choreography Description Language (WS-CDL) is an XML based language that describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal.

Policy constraints can be more easily checked for consistency against this model as it is more abstract than the behavioural implementations and uses a consistent language to describe the entire composition. It provides a global view of the complete composition rather than local views, allowing multiple levels of composition to be checked against the constraints.

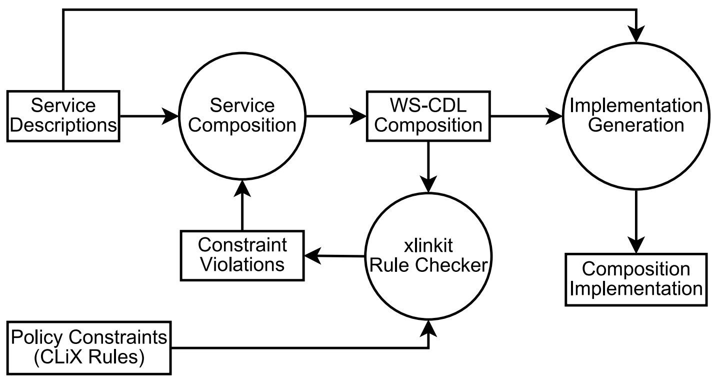


Fig. 2. Overview of policy constraint checking.

The diagram in figure 2 shows the process in which our approach is used. Service descriptions of existing services are used to create a WS-CDL composition. Policy constraints are written as CLiX rules. Both the policy constraints and the WS-CDL composition are input into the `xlinkit` rule checker. This results in a list of violations which describe how the WS-CDL composition violates the constraints. The violations are used as input to the service composition process so that the constraint violations can be corrected. This continues iteratively until the WS-CDL composition no longer violates any constraints and the implementation generation process can begin. This process can involve generating and manually writing code and results in the creation of an implementation of the service composition.

4 Communication Constraints

Using the CLiX language, we can describe constraints which limit the types of communication allowed between the roles in the choreography. We illustrate our approach using the example from the WS-CDL Primer[8]. This example involves four roles; a buyer, seller, credit agency and shipper. In this example, the buyer and seller negotiate until they agree on a price for an order. The buyer can then place the order. The seller performs a credit check, using the service provided by the credit agency, before placing a request with the shipper for the order to be dispatched. We have extended this example by completing some of the missing WS-CDL description which is not present in the primer.

Using the above example, a possible communication constraint is that a credit check should not be performed as part of the choreography, perhaps because a certain group of clients do not want these checks to be performed. Obviously, the example choreography described previously will violate this policy. The WS-CDL fragment which describes the credit check is shown in figure 3. This interaction involves the operation which performs the credit check, 'creditCheck', being performed on the 'CreditRole'.

```
<interaction channelVariable="tns:CreditAgencyC"
  name="CheckCredit" operation="creditCheck">

  <participate fromRoleTypeRef="tns:SellerRole"
           relationshipType="tns:Seller2Credit"
           toRoleTypeRef="tns:CreditRole"/>
  <exchange action="request"
            informationType="tns:CreditRequestType"
            name="CreditRequest">
    ...
  </exchange>
  <exchange action="respond"
            informationType="tns:CreditResponseType"
            name="CreditResponse">
     ...
  </exchange>
  <exchange action="respond" faultName="CreditCheckFault"
            informationType="tns:CreditFailureType"
            name="CreditFailure">
    ...
  </exchange>
  ...
</interaction>
```

Fig. 3. WS-CDL document fragment which performs a credit check.

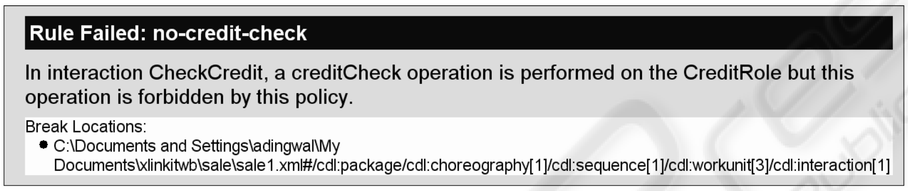
The CLiX rule which ensures that the policy constraint is satisfied is shown in figure 4. This rule requires that for any interaction where the target role of the interaction is the credit agency role, the `creditCheck` operation is not the operation which is being called. The outer `forall` element selects the set of all `interaction` elements in the WS-CDL document and for each one evaluates the enclosed rule. The enclosed rule must be satisfied for every interaction if the rule as a whole is to be satisfied. The `implies` element requires that if the first sub-rule is true then the second must be true . The first sub-rule is true when the `toRoleTypeRef` attribute of the `participate` element of an `interaction` element is the `CreditRole`. If this is the case then the second sub-rule requires that the `operation` attribute of the `interaction` element should not have the value `creditCheck`. One complication occurs due to the use of XML namespaces in the first sub-rule. As `xlinkit` uses XPath 1.0, it is necessary to select only the local part of the role reference by using the `substring-after` XPath function to select only the portion which occurs after the colon.

When this rule is applied to the WS-CDL document fragment in figure 3 it results in a constraint violation. Figure 5 shows the HTML output from `xlinkit`. This document includes a reference to the element which caused the constraint violation which specifies the file where the violation occurred and, using XPath, identifies the individual interaction element. This HTML document could be used as input to other tools which help resolve the violation.

```
<forall var="$interaction" in="//cdl:interaction">
  <implies>
    <equal op1="substring-after
                ($interaction/cdl:participate/@toRoleTypeRef, ':')"
           op2="'CreditRole'"/>

    <notequal op1="$interaction/@operation"
              op2="'creditCheck'"/>
  </implies>
</forall>
```

Fig. 4. CLiX rule which forbids a choreographies from performing ‘creditCheck’ interactions.



```
Rule Failed: no-credit-check
In interaction CheckCredit, a creditCheck operation is performed on the CreditRole but this
operation is forbidden by this policy.
Break Locations:
 ● C:\Documents and Settings\adingwal\My
   Documents\xlinkitwb\sale\sale1.xml#/cdl:package/cdl:choreography[1]/cdl:sequence[1]/cdl:workunit[3]/cdl:interaction[1]
```

Fig. 5. Output showing constraint violations.

5 Constraints on Interaction Ordering

The communication constraints described in the previous section can be expressed in `xlinkit` in a fairly straightforward manner. Another type of constraint which we would like to check are constraints on the order of interactions. For example, a credit check interaction must occur before shipping is requested. Such a constraint is not easy to express using the standard `xlinkit` operators. To make this easier, we have created a custom operator for determining whether an interaction in a WS-CDL document occurs before another.

In WS-CDL, three main structures affect ordering. The ‘sequence’ structure requires that the enclosed activities must be performed in the specified order. The ‘parallel’ structure requires that all the enclosed activities must be performed but places no restrictions on ordering and allows simultaneous execution. The ‘choice’ structure requires that exactly one of the enclosed activities is performed. The other important construct which affects ordering is the ‘workunit’ construct. This allows recursion as well as preventing the enclosed condition from being executed until a guard condition is satisfied.

Our operator determines ordering using basic information available by static analysis of the WS-CDL document. Importantly, it does not consider the value of variables, even if it is possible to determine the value of those variables statically. The operator is pessimistic, returning a false result unless it is able to determine with certainty that the ordering constraint is satisfied. In general this should be sufficient as static ordering constraints should be imposed using constructs which do not depend on variables. While it is possible to create ordering which depends on the values of variables but can actually be determined statically, we do not detect these cases, and we consider this something to be avoided when creating WS-CDL documents.

The ordering structures in WS-CDL allow the following rules to be applied. After a ‘sequence’ or a ‘parallel’ construct has completed execution, all the actions contained in these constructs have completed. After a ‘choice’ construct is complete, it is not possible to know which of the possible actions will have been completed. A construct is executing if it is an ancestor of the current interaction. During execution of a ‘sequence’ structure, earlier actions in the sequence have completed while later one have not. It is not possible to know which actions have completed during execution of a ‘parallel’ or ‘choice’ construct.

Using these rules, a recursive algorithm was written which traverses the tree structure of ordering elements. The algorithm ascends the tree looking for previous interactions which match the required interaction name. When it reaches a sequence element, the algorithm is able to descend the tree and search previous interactions, also descending down parallel elements and further sequence elements. The recursion terminates when the enclosing choreography element is reached. The algorithm is also able to understand ‘perform’ elements which cause sub-choreographies in the same document to be performed. The algorithm follows the references to these sub-choreographies.

Figure 6 shows an example rule which enforces the constraint that a ‘CreditCheck’ interaction should occur before a ‘RequestShipping’ interaction, using the ‘before’ operator. It requires that for all ‘RequestShipping’ interactions there exists a ‘CheckCredit’ interaction which occurs before it in the WS-CDL choreography.

```
<forall var="shippingInteraction"
       in="//cdl:choreography//cdl:interaction[@name='RequestShipping']">
  <exists var="creditCheckInteraction"
          in="//cdl:choreography//cdl:interaction[@name='CheckCredit']">
    <operator name="order:before">
      <param name="a" value="$creditCheckInteraction"/>
      <param name="b" value="$shippingInteraction"/>
    </operator>
  </exists>
</forall>
```

Fig. 6. Rule which requires that a ‘CheckCredit’ interaction should occur before a ‘RequestShipping’ interaction.

Figure 7 shows how the ‘before’ operator is applied to a WS-CDL document. This diagram represents the structure of fragment of a WS-CDL document, showing a choreography element with a number of ordering elements and interaction elements. Arrows show how the algorithm traverses the document tree to search for an interaction which matches the ordering constraint. Starting from the ‘RequestShipping’ element, the algorithm ascends the tree and encounters a sequence element. This action is not yet completed as the algorithm ascended the tree to reach it, so it is only necessary to search earlier actions in the sequence. The algorithm is able to descend to the ‘parallel’ element but not to the interaction ‘E’ which occurs later in the sequence. The ‘parallel’ element is a completed action, as the algorithm reached it by descending the tree, so

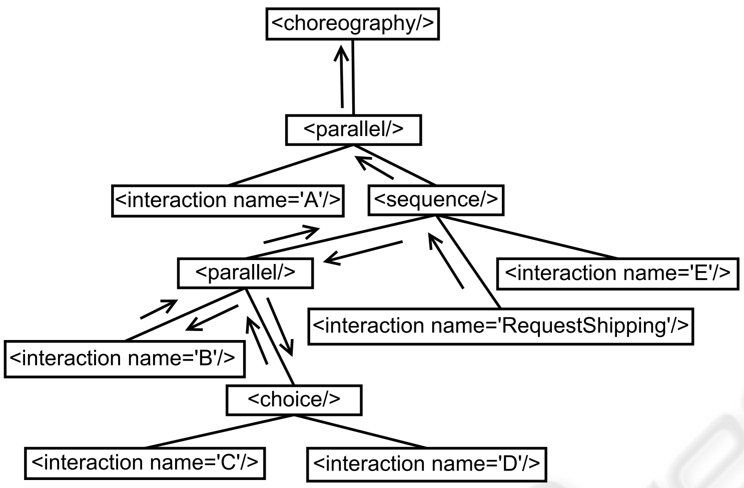


Fig. 7. An example showing how the algorithm used by the 'before' operator is applied to a WS-CDL document.

all child elements of this element can be searched (this would also be the case if the element was a sequence element). Descending to the 'choice' element, the algorithm cannot descend further as it is not possible to tell which of the child actions will be performed. The algorithm then descends to the interaction element 'B' and checks to see if this interaction has the name 'CheckCredit'. As it does not, the algorithm continues by ascending the document tree. From the sequence element, the algorithm ascends to another parallel element. As this action has not completed, it is not necessary to descend to interaction A. The algorithm ascends again to the choreography element at which point the algorithm completes. In this case, a matching interaction was not found so the operator returns the result false.

6 Related Work

The work described in this paper builds on[9] in which `xlinkit` was also used to check consistency with policy constraints. The principle difference is that the previous work checked WSDL documents against constraints. WSDL documents describe an interface but give little information about how the interface is used or how different services are composed. Furthermore, WSDL descriptions are supplied by service providers but policy constraints need to be obeyed by service consumers who are responsible for the composition. In contrast, WS-CDL documents describe interactions between different services and the ordering of those interactions, allowing more complex policy constraints to be checked.

7 Conclusions and Future Work

In this paper we have described an approach for checking policy constraints for compositions of web services described in WS-CDL. Policy constraints are expressed as CLiX rules on the WS-CDL choreography. These constraints can be evaluated using `xlinkit` which identifies constraint violations in the choreography. Our contribution is a lightweight approach to the problem of checking that policy constraints are obeyed by web service compositions, using existing technology. The approach is implemented using the existing `xlinkit` rule checking engine.

We have been able to specify constraints relating to how different roles in a web service composition can communicate with each other and in the ordering of interactions. Future work may include identifying further types of constraints which can be checked using this approach and whether further CLiX operators need to be defined to support them. Other constraints which might be of interest include constraints related to issues such as reliability and performance. Unfortunately, WS-CDL does not describe such properties making it impossible to check constraints related to these issues. This problem could be solved by extending the WS-CDL description with these properties, allowing constraints which use these extensions to be described.

A potential problem with our consistency rules is that they rely on the the names used in the WS-CDL files being consistent with those used in the CLiX rules. If the 'CreditAgency' role is referred to as 'CreditReferenceAgancy' in a particular WS-CDL file then rule which refers to the 'CreditAgency' role will fail. To write meaningful rules, WS-CDL documents and CLiX rule documents must share a common taxonomy. This need not necessarily be formally specified, as long as all documents use the same names to refer to the same entities. This requirements could also be enforced to some extent using CLiX rules which constrain WS-CDL documents to use only names which are present in a particular taxonomy.

Our approach checks policy constraints are consistent with WS-CDL models rather than with the actual implementation of web service compositions. To show that the actual web service composition is consistent with the constraints, it is necessary to ensure that the implementation of the composition is consistent with the WS-CDL model. There are two approaches to this problem. The first is to generate implementation from the model, including WSDL interfaces, BPEL orchestrations and general purpose language code. This ensures that generated implementation code is consistent with the model at the time of generation. Another approach might be to check consistency between the model and implementation. This has the advantage that existing implementations can be checked and that inconsistencies which are introduced subsequent to code generation can be detected. For example, it may be useful to identify consistency rules between WS-CDL and WSDL interfaces.

Acknowledgements

This work has been partially sponsored by the EU project SENSORIA, IST-2005-016004.

References

1. W3C: Web services choreography description language version 1.0. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/ (2005)
2. Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A.: xlinkit: A consistency checking and smart link generation service. ACM Transactions on Internet Technology **2** (2002) 151–185
3. Nentwich, C., Emmerich, W., Finkelstein, A., Ellmer, E.: Flexible consistency checking. ACM Transactions on Software Engineering and Methology **12** (2003) 28–63
4. Nentwich, C., Emmerich, W., Finkelstein, A.: Consistency management with repair actions. In: the 25th International Conference on Software Engineering, IEEE Press (2003) 455–464
5. Marconi, M., Nentwich, C.: Clix language specification version 1.0. http://www.clixml.org/clix/1.0/clix.xml (2004)
6. W3C: Xml path language (xpath) version 1.0. http://www.w3.org/TR/xpath (1999)
7. ECMA: Ecmascript language specification. http://www.ecma-international.org/publications/standards/Ecma-262.htm (1999)
8. W3C: Web services choreography description language: Primer. http://www.w3.org/TR/ws-cdl-10-primer (2006)
9. Piccinelli, G., Finkelstein, A., Nentwich, C.: Web services need consistency. In: OOPSLA 2002 Workshop on Object-Oriented Web Services. (2002)