Rules-based SSDL Protocol Framework

Dean Kuo[1], Paul Greenfield[1], Savas Parastatidis[2], Jim Webber[3]
Dean.Kuo@csiro.au, Paul.Greenfield@csiro.au, Savas@Parastatidis.name, Jim@Webber.name

Abstract

The Rules-based SSDL Protocol Framework defines a collection of XML Infoset element information items that can be used to describe the messaging behaviour of a service. The framework use conditions (Boolean expressions) to precisely specify when messages can be sent and received by a service. A distributed application's application protocol is defined by the messaging behaviour of interacting services participating in the distributed application.

SSDL documents

An Introduction to the SOAP Service Description Language [1]

SOAP Service Description Language (SSDL) v1.3 [2]

MEP SSDL Protocol Framework v1.3 [3]

CSP SSDL Protocol Framework v1.3 [4]

Rules SSDL Protocol Framework v1.3 [5] (this document)

Sequencing Constraints SSDL Protocol Framework v1.3 [6]

Status of this document

Version: 1.3

Date: April 2005

http://ssdl.org

Disclaimer

[1] CSIRO Information and Communication Technology (ICT) Centre, CSIRO, Australia
[2] School of Computing Science, University of Newcastle, Newcastle upon Tyne, NE1 7RU, UK
[3] ThoughtWorks Australia Pty. Ltd.

Table of Contents

1. Introduction

The Rules-based SSDL protocol framework defines the structure and semantics of elements that can be used to describe a service's messaging behaviour using conditions (Boolean Expressions) [4]. A distributed application's application protocol (messaging protocol) is defined by the messaging behaviour of interacting services participating in the distributed application. Using the Rules-based SSDL protocol framework, model checkers can validate both safety and liveness properties of an application protocol.

1.1. SSDL

The SOAP Service Description Language (SSDL) is a SOAP-centric contract description language for Web Services. It is meant as a means for exploring ideas in the areas of contract and protocol description and Web Services implementations using message-oriented programming abstractions.

1.1.1. Motivation

SOAP is the standard message transfer protocol for Web Services. However, the default description language for Web Services (WSDL) does not explicitly target SOAP but, instead, provides a generic framework for the description of network-exposed software artefacts. The work on SSDL aims to investigate the advantages/disadvantages of Web Services description when SOAP is assumed from the outset compared to the transfer-independent approach of WSDL. The use of formal models for describing message-based interactions is also a goal for SSDL. Finally, this work aims to demonstrate the benefits of focusing on message-orientation when architecting, designing, and building Web Services rather than on the interface and remote procedure call abstractions.

1.1.2. The Language

The SOAP Service Description Language provides the base framework for a range of protocol description frameworks which at one end of the spectrum can be a simpler, SOAP-focussed, direct replacement for WSDL MEPs while at the other end of the spectrum can enable formal validation and reasoning about the protocols that a Web Service supports.

The frameworks are componentised in a similar way to the WS-Policy suite of specifications, with a base SSDL framework providing the fundamental protocol building blocks for describing messages while other specifications utilise those building blocks to describe the way in which the messages participate in the protocols that a Web Service supports. This is illustrated in Figure 1.

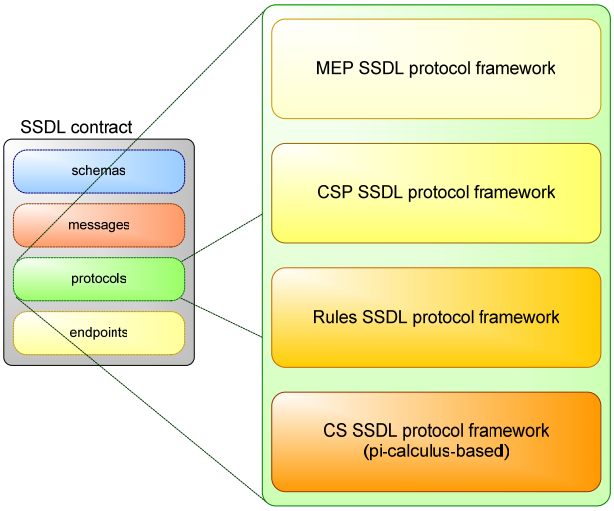

Figure 1 The SSDL Suite of Specifications

Four protocol description frameworks are provided with the base SSDL specification but it is expected that others will be implemented to meet different needs. The protocol description frameworks provided with the initial release of SSDL 1.0 are:

- MEP - The Message Exchange Patterns (MEP) Framework defines a collection of XML Infoset element information items that represent commonly used simple exchange patterns. The current set of message exchange patterns supported by the MEP framework is a superset of that found in the latest draft of the WSDL specification.
- CSP – The Communicating Sequential Processes (CSP) Framework defines a collection of XML Infoset element information items for defining a multi-message exchange using sequential process semantics, based on basic CSP semantics. Work is currently underway to make use of the full strength of the CSP in describing contracts.
- Rules - The Rules-based SSDL Protocol Framework defines a collection of XML Infoset element information items that can be used to describe a multi-message exchange protocol using conditions. The protocols captured using the Rules-based SSDL protocol framework can be validated for correctness, liveness, and other properties.
- SC - The Sequencing Constraints (SC) Protocol Framework defines a collection of XML Infoset element information items that can be used to describe a multi-party, multi-message exchange protocol using notations based on the pi-calculus. Protocols in the framework are specified using a sequential technique, specifying the legal set of actions at each stage of the protocol. The framework is intended to provide a simple way of specifying protocols but also have a formal basis to allow properties of the protocols to be determined.

1.1.3. Key Features

- SSDL assumes SOAP as the means of transferring messages between Web Services over arbitrary transport (and transfer) protocols. It has been designed to work harmoniously with all aspects of the underlying SOAP processing model. As a result, there is no need to define bindings for all possible transport protocols;
- SSDL assumes WS-Addressing as the standard means for embedding addressing information within SOAP envelopes and for binding those addresses onto underlying transport protocols;
- SSDL focuses on messages and protocols. As a result, there is no need for articles like ‘interface’, ‘inheritance’, and ‘operation’;
- XML Infoset is assumed as the underlying SSDL component model. There is no need (nor desire) to create a new component model simply for contract description;
- Modularisation of contracts is handled using XInclude. A shortcut mechanism is provided which is defined in terms of XInclude elements to simplify componentisation as far as is possible;
- SSDL promotes protocol framework extensibility. It allows different protocol description models to be plugged into the base SSDL framework which helps promote protocol-based integration and exposure of the messaging behaviour of a Web Service. Tools such as model checkers can verify the correctness of protocols defined in an SSDL contract, or automate the reasoning about the compatibility of Web Services. Hosting environments can even use the SSDL contract to validate the message exchanges between Web Services.

1.2. Goals

- To facilitate a succinct method that can fully specify the messaging behaviour of a SOAP Web Service. The method can specify the incoming and outgoing messages of a service, their causal relationships and the asynchronous interactions between services.
- To enable automatic checking for safety and liveness properties of a messaging protocol defined by the contracts of interacting services.

1.3. Example

The contract shown below describes some of the messages sent and received by a merchant service to process payment from a customer service. The messages include invoice, payment and receipt messages as well as messages for dealing with late payment and payment faults.

```
<?xml version="1.0" encoding="utf-8" ?>
<ssdl:contract targetNamespace="http://example.org/service/contract"
               xmlns:ssdl="urn:ssdl:v1">
  <ssdl:schemas>
    <!-- schemas -->
  </ssdl:schemas>

  <ssdl:messages targetNamespace="http://example.org/service/messages"
                 xmlns:tns="http://exaxmple.org/service/schema.xsd">

    <ssdl:message name="PurchaseOrderMsg">
      <ssdl:body ref="tns:PurchaseOrderDescription"/>
    </ssdl:message>

    <ssdl:message name="InvoiceMsg">
      <ssdl:body ref="tns:InvoiceDescription"/>
    </ssdl:message>

    <ssdl:message name="PaymentMsg">
      <ssdl:body ref="tns:PaymentDescription"/>
    </ssdl:message>

    <ssdl:message name="ReceiptMsg">
      <ssdl:body ref="tns:ReceiptDescription"/>
    </ssdl:message>

    <ssdl:fault name="PaymentFault">
      <ssdl:code value="Sender"/>
    </ssdl:fault>

    <ssdl:message name="LateFeeMsg">
      <ssdl:body ref="tns:LateFeeDescription"/>
    </ssdl:message>
  </ssdl:messages>

  <ssdl:protocols>
    <ssdl:protocol targetNamespace="http://example.org/service/protocol"
                   xmlns:msgs="http://example.org/service/messages"
                   xmlns:rls="urn:ssdl:rules:v1">
      <rls:rules>

        <rls:rule>                                                          (1)
         <!-- rule -->
        </rls:rule>
          …
        <rls:rule>
          <!-- rule -->
        </rls:rule>

        <rls:rule>                                                          (2)
          <ssdl:msgref ref="InvoiceMsg" direction="out" />
          <rls:condition>
            <rls:and>
              <ssdl:msgref ref="PurchaseOrderMsg" direction="in" />
              <rls:not>
                <ssdl:msgref ref="InvoiceMsg" direction="out" />
              </rls:not>
            </rls:and>
```

```
          </rls:condition>
        </rls:rule>

        <rls:rule>                                                              (3)
          <ssdl:msgref ref="PaymentMsg" direction="in" />
          <rls:condition>
            <rls:and>
              <ssdl:msgref ref="InvoiceMsg" direction="out" />
               <rls:not>
                 <ssdl:msgref ref="PaymentMsg" direction="in" />
               </rls:not>
            </rls:and>
          </rls:condition>
        </rls:rule>

        <rls:rule>                                                              (4)
          <ssdl:msgref ref="ReceiptMsg" direction="out" />
          <rls:condition>
            <rls:and>
              <ssdl:msgref ref="PaymentMsg" direction="in" />
              <rls:not>
                <ssdl:msgref ref="ReceiptMsg" direction="out" />
              </rls:not>
            </rls:and>
          </rls:condition>
        </rls:rule>

        <rls:rule>                                                              (5)
           <ssdl:msgref ref="PaymentFault" direction="out" />
           <rls:condition>
             <rls:and>
               <ssdl:msgref ref="PaymentMsg" direction="in" />
               <rls:not>
                 <ssdl:msgref ref="ReceiptMsg" direction="out" />
                 <ssdl:msgref ref="PaymentFault" direction="out"/>
               <rls:not>
             </rls:and>
           </rls:condition>
         </rls:rules>

        <rls:rule>                                                              (6)
          <ssdl:msgref ref="LateFeeMsg" direction="out" />
          <rls:condition>
            <rls:and>
              <ssdl:msgref ref="InvoiceMsg" direction="out" />
              <rls:not>
                <ssdl:msgref ref="LateFeeMsg" direction="out" />
                <ssdl:msgref ref="ReceiptMsg" direction="out" />
              </rls:not>
            </rls:and>
          </rls:condition>
        </rls:rule>

   </ssdl:protocol>

   <ssdl:endpoints>
     <ssdl:endpoint xmlns:wsa="http://www.w3.org/2004/12/addressing">
       <wsa:Address>http://example.org/service</wsa:Address>
     </ssdl:endpoint>
   </ssdl:endpoints>
</ssdl:contract>
```

Example 1: A contract with a protocol defined using the Rules SSDL Protocol Framework

In Example 1, the rules-based SSDL protocol framework expresses the following message exchange pattern is supported by a merchant service. It is assumed that we use reliable messages to deliver messages, messages are delivered in FIFO order with unbounded message latency. The rules defined in this example are:

(1) Defines the messaging behaviour prior to payment. They include receiving a quote request, sending a quote and receiving a purchase order.

(2) An `InvoiceMsg` message can only be sent if it has not been sent previously and the merchant has received a `PurchaseOrderMsg`. Once a merchant receives a purchase order from the customer, it can then send an invoice to initiate payment.

(3) A `PaymentMsg` message can be received after the `InvoiceMsg` has been sent and a `PaymentMsg` has not already been received. A merchant can only expect to receive payment after it has sent an invoice.

(4) A `ReceiptMsg` message can be sent after the `PaymentMsg` has been received and a `ReceiptMsg` has not already been sent. Once payment has been received, a receipt can be sent.

(5) A `PaymentFault` message can be sent only after receiving a `PaymentMsg message`, and only if neither a `ReceiptMsg` nor a `PaymentFault` have already been sent. A receipt acknowledges a successful payment. The rule specifies the service will not send a payment fault message after sending the receipt. Rules (4) and (5) combined specify that the merchant service can respond with either a `ReceiptMsg` message or a `PaymentFault` message.

(6) A `LateFeeMsg` message can be sent after an `InvoiceMsg` has been sent and a `ReceiptMsg` message has not been sent. Similar to the payment fault message, the rule specify that once the merchant has sent a receipt, it will not send a late fee message.

The above example describes, partially, the messaging behaviour of a merchant service participating in an application protocol. There may be race issues with the messages sent and received (e.g. the `PaymentMsg` may be in transit when the `LateFeeMsg` is sent), deadlock and incorrect termination when interacting with a customer service. It is the responsibility of the application developer to ensure the correctness of multi-message protocol arising from the possible interactions between such services. Correctness means that the protocol is free from deadlocks and race conditions, and terminates correctly. It is possible [4] to use tools such as model checkers and design patterns to aid developers in the design of correct application protocols.

1.4. Framework Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [7].

This specification uses properties from the XML Information Set [8]. Such properties are denoted by square brackets and in bold, e.g. **[namespace name]**.

This specification uses namespace prefixes throughout; they are listed in Table 1-1. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [8]).

We use the pseudo-schema notation used in WSDL 2.0 Core [9] as a convenient description of the structure of a component.

1.5. Namespaces

These namespaces and their prefixes are used throughout this document.

Prefix	Namespace	Notes
ssdl	urn:ssdl:v1	
xs	http://www.w3.org/2001/XMLSchema	
rls	urn:ssdl:rules:v1	Where elements are not qualified with a namespace prefix, urn:ssdl:rules:v1 is assumed

2. General Description

The Rules framework for SSDL provides a means of expressing relationships between messages defined in a SSDL contract for a Web Service. These relationships are specified by conditions (Boolean expressions) that precisely specify when a message can be sent and received. These conditions are defined only in terms of what message have and have not been sent and received, and forms a description of the valid flow of messages into and out of a service. It describes a service's messaging behaviour.

A service's messaging behaviour specified in the SSDL rules framework can be translated into model checker code. Model checkers can take the messaging behaviours of interacting services and verify the correctness (safety and liveness properties) of the resulting application protocol.

3. Protocol Framework Structure

This section describes the Rules SSDL Protocol Framework structure using the XML Information Set [8] model.

```
<ssdl:protocol>
  <rules>
    <rule /> +
  </rules>
</ssdl:protocol>
```

The Rules SSDL protocol framework defines the following element information items which can be used within the `ssdl:protocol` element information item of an SSDL contract:

- A REQUIRED `rules` element information item with the following properties:
  - A **[local name]** of "rules"
  - A **[namespace name]** of "urn:ssdl:rules:v1"
  - One or more `rule` element information items in the **[children]** property

3.1. rule

```
<ssdl:protocol>
  <rules>
    <rule>
      <ssdl:msgref/> +
      <condition />
    <rule>
  </rules>
</ssdl:protocol>
```

The `rule` element information item defines a rule for a message. It has the following properties:

- A **[local name]** of "rule"
- A **[namespace name]** of "urn:ssdl:rules:v1"
- A REQUIRED `condition` element information item in the **[children]** property
- One or more `ssdl:msgref` element information item in the **[children]** property

The `ssdl:msgref` element information items refer to the messages and their direction to which the defined rule applies.

The protocol framework specifies precisely when messages can and cannot be received by a service. For incoming messages, only when the `condition` element evaluates to true can the message be received by the service. For outgoing messages, only when the `condition` element evaluates to true can the message be sent by the service.

3.1.1. condition

```
<ssdl:protocol>
```

```
  <rules>
    <rule>
      <condition>
        [ <ssdl:msgref /> |
          <and /> |
          <or /> |
          <xor /> |
          <not /> ]
      </condition>
    <rule>
  </rules>
</ssdl:protocol>
```

The `condition` element information item represents the Boolean expression that is associated with the rule. Only when the Boolean expression evaluates to true, the rule is considered active. The `condition` element information item has the following properties:

- A **[local name]** of "condition"
- A **[namespace name]** of "urn:ssdl:rules:v1"
- Zero or more of the following element information items in its **[children]** property:
    - An `and` element information item
    - An `or` element information item
    - An `xor` element information item
    - A `not` element information item
    - A `ssdl:msgref` element information item
- If the `condition` element has no element information items in its **[children]** property then the `condition` evaluates to true.

3.1.1.1 ssdl:msgref as a Boolean expression

An `ssdl:msgref` element information item is considered to have the Boolean value "true" if and only if the message has been

- received if the **[normalised value]** of its `direction` attribute information item is "in", or
- sent if the **[normalised value]** of its `direction` attribute information item is "out".

3.1.1.2 and

```
<ssdl:protocol>
  <rules>
    <rule>
      <condition>
        <and>
          [ <ssdl:msgref /> |
            <and /> |
            <or /> |
            <xor /> |
            <not /> ] +
        </and>
      </condition>
    <rule>
  </rules>
</ssdl:protocol>
```

The `and` element information item represents a logical "AND" Boolean expression on the Boolean expressions represented by its **[children]** element information item. The `and` element information item has the following properties:

- A **[local name]** of "and"
- A **[namespace name]** of "urn:ssdl:rules:v1"

- One of the following element information items in its **[children]** property:
    - An `ssdl:msgref` element information item
    - An `and` element information item
    - An `or` element information item
    - An `xor` element information item
    - A `not` element information item

3.1.1.3 or

```
<ssdl:protocol>
  <rules>
    <rule>
      <condition>
        <or>
          [ <ssdl:msgref /> |
            <and /> |
            <or /> |
            <xor /> |
            <not /> ] +
        </or>
      </condition>
    <rule>
  </rules>
</ssdl:protocol>
```

The `or` element information item represents a logical "OR" Boolean expression on the Boolean expressions represented by its **[children]** element information item. The `or` element information item has the following properties:

- A **[local name]** of "or"
- A **[namespace name]** of "urn:ssdl:rules:v1"
- One of the following element information items in its **[children]** property:
    - An `ssdl:msgref` element information item
    - An `and` element information item
    - An `or` element information item
    - An `xor` element information item
    - A `not` element information item

3.1.1.4 xor

```
<ssdl:protocol>
  <rules>
    <rule>
      <condition>
        <xor>
          [ <ssdl:msgref /> |
            <and /> |
            <or /> |
            <xor /> |
            <not /> ] +
        </xor>
      </condition>
    <rule>
  </rules>
</ssdl:protocol>
```

The `xor` element information item represents a logical "XOR" Boolean expression on the Boolean expressions represented by its **[children]** element information item. The `xor` element information item has the following properties:

- A **[local name]** of “xor”
- A **[namespace name]** of “urn:ssdl:rules:v1”
- One of the following element information items in its **[children]** property:
    - An `ssdl:msgref` element information item
    - An `and` element information item
    - An `or` element information item
    - An `xor` element information item
    - A `not` element information item

3.1.1.5 not

```
<ssdl:protocol>
  <rules>
    <rule>
      <condition>
        <not>
          [ <ssdl:msgref /> |
            <and /> |
            <or /> |
            <xor /> |
            <not /> ] +
        <not>
      </condition>
    <rule>
  </rules>
</ssdl:protocol>
```

The `not` element information item represents a logical “NOT” Boolean expression on all the Boolean expressions represented by its **[children]** element information item. The `not` element information item has the following properties:

- A **[local name]** of “not”
- A **[namespace name]** of “urn:ssdl:rules:v1”
- One of the following element information items in its **[children]** property:
    - An `ssdl:msgref` element information item
    - An `end` element information item
    - An `or` element information item
    - An `xor` element information item
    - A `not` element information item

A `not` element information item that has more than one element information items in its **[children]** property is equivalent to a `not` element information item being the **[parent]** of each one of them and all of the `not` element information items being **[children]** of an `and` element information item. For example:

```
<not>
  <msgref ref="a"/>
  <msgref ref="b"/>
  <or>
    <msgref ref="c"/>
    <msgref ref="d"/>
  </or>
  <msgref ref="e"/>
</not>
```

Is equivalent to

```
<and>
  <not>
    <msgref ref="a"/>
  </not>
  <not>
    <msgref ref="b"/>
  </not>
  <not>
    <or>
      <msgref ref="c"/>
      <msgref ref="d"/>
    </or>
  </not>
  <not>
    <msgref ref="e"/>
  </not>
</and>
```

3.2. final attribute for ssdl:msgref

The **[normalised value]** of the `final` attribute information item is a Boolean (xs:Boolean) that indicates whether the referred message is the last one in the protocol. The `final` attribute information can only appear in the **[attributes]** property of an `ssdl:msgref` element information item.

The `final` attribute information item has the following properties:

- A **[local name]** of “final”
- A **[namespace name]** of “urn:ssdl:rules:v1”

References

[1] S. Parastatidis, J. Webber, S. Woodman, D. Kuo, and P. Greenfield, "An Introduction to the SOAP Service Description Language," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-898, 2005.

[2] S. Parastatidis, J. Webber, S. Woodman, D. Kuo, and P. Greenfield, "SOAP Service Description Language (SSDL)," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-899, 2005.

[3] S. Parastatidis and J. Webber, "MEP SSDL Protocol Framework," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-900, 2005.

[4] S. Parastatidis and J. Webber, "CSP SSDL Protocol Framework," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-901, 2005.

[5] D. Kuo, S. Parastatidis, and J. Webber, "Rules SSDL Protocol Framework," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-902, 2005.

[6] S. Woodman, S. Parastatidis, and J. Webber, "Sequencing Constraints SSDL Protocol Framework," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-903, 2005.

[7] S. Bradner, "IETF RFC 2119: Key words for use in RFCs to Indicate Requirement Levels." http://www.ietf.org/rfc/rfc2119.txt: Internet Engineering Task Force, 1999.

[8] W3C, "XML Information Set." http://www.w3.org/TR/xml-infoset/, 2004.

[9] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer, and S. Weerawarana, Eds. http://www.w3.org/TR/2004/WD-wsdl20-20040803/, 2004.

Appendix A – XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
    $Modtime: 14/02/05 9:25 $
```

```
    $Revision: 13 $
-->
<xs:schema
      elementFormDefault="qualified"
      targetNamespace="urn:ssdl:rules:v1"
      xmlns:ssdl="urn:ssdl:v1"
      xmlns:rls="urn:ssdl:rules:v1"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:import namespace="urn:ssdl:v1" />

   <xs:element name="rules">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="rule" type="rls:rule-type"
                        minOccurs="1" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:complexType name="rule-type">
      <xs:sequence>
         <xs:element ref="ssdl:msgref" minOccurs="1" maxOccurs="unbounded" />
         <xs:element name="condition" type="rls:condition-type"
                     minOccurs="1" maxOccurs="1" />
      </xs:sequence>
   </xs:complexType>

   <xs:attribute name="final" type="xs:boolean" default="false"/>

   <xs:complexType name="condition-type">
      <xs:choice minOccurs="0" maxOccurs="1">
         <xs:element ref="ssdl:msgref" />
         <xs:element name="and" type="rls:operator-type" />
         <xs:element name="or" type="rls:operator-type" />
         <xs:element name="xor" type="rls:operator-type" />
         <xs:element name="not" type="rls:operator-type" />
      </xs:choice>
   </xs:complexType>

   <xs:complexType name="operator-type">
      <xs:choice minOccurs="1" maxOccurs="unbounded">
         <xs:element ref="ssdl:msgref" />
         <xs:element name="and" type="rls:operator-type" />
         <xs:element name="or" type="rls:operator-type" />
         <xs:element name="xor" type="rls:operator-type" />
         <xs:element name="not" type="rls:operator-type" />
      </xs:choice>
   </xs:complexType>
</xs:schema>
```