Sequencing Constraints SSDL Protocol Framework

Simon Woodman[1], Savas Parastatidis[1], Jim Webber[2]
S.J.Woodman@ncl.ac.uk, Savas@Parastatidis.name, Jim@Webber.name

Abstract

The Sequencing Constraints (SC) SSDL 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.

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]

Sequencing Constraints SSDL Protocol Framework v1.3 [6] (this document)

Status of this document

Version: 1.3

Date: April 2005

http://ssdl.org

Disclaimer

[1] School of Computing Science, University of Newcastle, Newcastle upon Tyne, NE1 7RU, UK
[2] ThoughtWorks Australia Pty. Ltd.

Table of Contents

1. Introduction

The Sequential Constraint (SC) SSDL Protocol Framework provides a machine-readable description which is used to define the conversations that a Web Service supports. Such conversations may be a set of request-response interactions or could use several messages involving multiple parties over arbitrary lengths of time. The framework is intended to provide a simple way of specifying such protocols but also have a formal basis to allow properties of the protocols to be determined if required. Protocols in the framework are specified using a sequential technique, specifying the legal set of actions at each stage of the protocol. It is believed that this leads to a description which is easy to understand, as at each step of the protocol the set of actions allowed is explicitly described. The SC SSDL protocol has a formal basis in the pi-calculus, a process algebra for describing mobile, communicating processes. The formal basis allows protocols described in the SC framework to be validated to ensure properties such as correctness, liveness and consistency.

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.

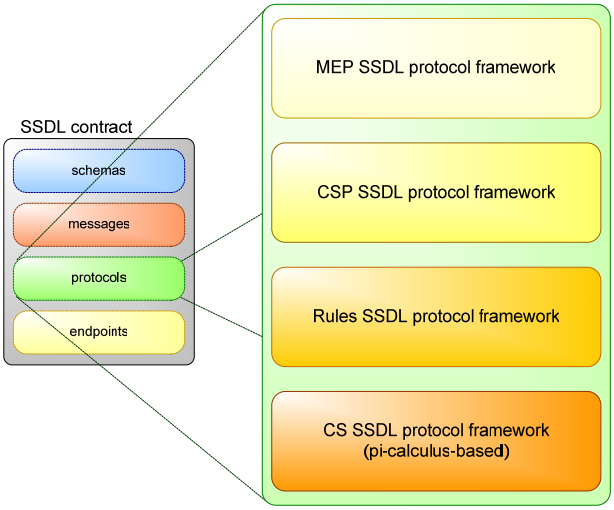

Figure 1 The SSDL Suite of Specifications

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 the description of conversations with SOAP Web Services
- To provide a convenient notation for description of conversations that is simple to understand
- To enable automatic checking for properties (correctness, liveness, consistency, etc) of a protocol supported by a service
- To allow compatibility checking between the protocols that two services support

1.3. Example

```
<?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:formats="http://exaxmple.org/service/schema.xsd">

   <ssdl:message name="purchase-order">
     <ssdl:body ref="formats:purchase-order-type"/>
   </ssdl:message>

   <ssdl:message name="item-not-available">
     <ssdl:body ref="formats:item-not-available-type"/>
   </ssdl:message>

   <ssdl:message name="purchase-order-ack">
     <ssdl:body ref="formats:purchase-order-ack-type"/>
   </ssdl:message>

   <ssdl:message name="cancel-order">
     <ssdl:body ref="formats:cacel-order-type"/>
   </ssdl:message>

   <ssdl:message name="cancel-order-ack">
     <ssdl:body ref="formats:cacel-order-ack-type"/>
   </ssdl:message>

   <ssdl:message name="confirm-order">
     <ssdl:body ref="formats:comfirm-order-type"/>
   </ssdl:message>

   <ssdl:message name="invoice">
     <ssdl:body ref="formats:invoice-type"/>
   </ssdl:message>

  </ssdl:messages>

  <ssdl:protocols>
    <ssdl:protocol targetNamespace="http://example.org/service/protocol"
                   xmlns:msgs="http://example.org/service/messages"
```

```
                  xmlns:sc="urn:ssdl:sc:v1">
      <sc:sc>
        <sc:participant name="purchaser"/>

        <sc:protocol name="process-purchase-order">
          <sc:sequence>
          <ssdl:msgref ref="msgs:purchase-order" direction="in" sc:participant="purchaser"/>
            <sc:choice>
              <ssdl:msgref ref="msgs:item-not-available" direction="out"
                           sc:participant="purchaser"/>
              <sc:sequence>
                <ssdl:msgref ref="msgs:purchase-order-ack" direction="out"
                             sc:participant="purchaser"/>
                <sc:choice>
                  <sc:sequence>
                    <ssdl:msgref ref="msgs:cancel-order" direction="in"
                                 sc:participant="purchaser"/>
                    <ssdl:msgref ref="msgs:cancel-order-ack" direction="out"
                                 sc:participant="purchaser"/>
                  </sc:sequence>
                  <sc:sequence>
                    <ssdl:msgref ref="msgs:confirm-order" direction="in"
                                 sc:participant="purchaser"/>
                    <ssdl:msgref ref="msgs:invoice" direction="out"
                                 sc:participant="purchaser"/>
                  </sc:sequence>
                </sc:choice>
              </sc:sequence>
            </sc:choice>
          </sc:sequence>
        </sc:protocol>
      </sc:sc>
    </ssdl:protocol>
  </ssdl:protocols>

  <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>
```

Listing 1: A contract with a protocol defined using the SC SSDL Protocol Framework

In Listing 1, the SC SSDL Protocol Framework used allows the specification of a simple multi-message protocol used by a retailer to process purchase orders from customers. The protocol begins by a participant named `purchaser` sending a `purchase-order` message to the retailer web service. Following this message, the retailer can respond either with an `item-not-available` message if the item is unknown or out of stock, or can agree to supply the goods by sending a `purchase-order-ack` message. If the fault message is sent the protocol terminates, whereas if the `puchase-order-ack` message is sent the protocol continues. The purchaser may now decide to buy the goods by sending a `confirm-order` message and then receiving an `invoice` message, or cancel the order by sending a `cancel-order` message and receiving a `cancel-order-ack` message. In both cases the protocol now terminates.

1.4. Relationship to the pi-calculus

The SC SSDL protocol framework has a formal basis in the pi-calculus (a calculus for mobile concurrent systems). A pi-calculus system is described in terms of processes, channels and names. Processes are independent and can communicate using channels that connect them. Each channel is referred to by a name and the communication unit along a channel is a name. A name is the most primitive form of addressing in pi-calculus. Processes are built from the following action terms and operators.

Primitive	Syntax	Description
Sequence	`P . Q`	Process P followed by process Q

Summation (choice)	`P + Q`	Either process P or process Q
Parallel composition	`P | Q`	Process P and process Q in parallel
Replication	`! P`	An infinite number of process P in parallel
Send	`a <x>`	Send name x along channel named a
Receive	`b (y)`	Receive a message along channel b and bind it to the name y

Computation in the pi-calculus is defined by a set of reaction rules which specify how a process initially in one state, P can move into another state, P' in one computational step. Every computation step consists of communication between two terms (either in the same or a different process). A full description of the pi-calculus and its application is beyond the scope of this document but more details can be found in [7]. It is possible to transform the SC SSDL into a pi-calculus representation which can be analysed using the reaction rules. From this, liveness of the specified protocol can be determined [8].

1.5. 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 [9].

This specification uses properties from the XML Information Set [10]. 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 [10]).

We use the pseudo-schema notation used in WSDL 2.0 Core [11] as a convenient description of the structure of a component.

1.6. 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	
sc	urn:ssdl:sc:v1	Where elements are not qualified with a namespace prefix, urn:ssdl:sc:v1 is assumed

2. General Description

The SC Protocol Framework for SSDL provides a means of expressing the valid sequence of externally visible actions a service may perform. In this context, an action is either the sending of a message or the receiving of a message. The structure of the protocol is described in terms of which of these actions must occur in sequence, which are exclusive choices, which can occur in parallel and which can occur a multiple number of times. It is possible to nest these elements to an arbitrary level in order achieve the desired protocol structure. The structure of the ordering is such that after any action it is easy to determine the set of legal actions which may occur next.

Model checkers and other tools may use this information to reason about the correctness of protocols. Another important use of this framework is simply that it makes Web Services self-descriptive insofar as both message format (from SSDL) and valid conversations are described.

3. Protocol Framework Structure

This section describes the SC SSDL Protocol Framework structure using the XML Information Set [10] model.

```
<ssdl:protocol>
  <sc>
    <participant/> +
    <protocol> +
  </sc>
</ssdl:protocol>
```

The SC SSDL protocol framework MUST contain a `sc` element information item that is used to group the sequencing constraints together. The `sc` element information item with the following properties:

- A **[local name]** of "sc"
- A **[namespace name]** of "urn:ssdl:sc:v1"
- The following element information items in its **[children]** property in order:
  - One or more `participant` element information items
  - One or more `protocol` element information items

3.1. participant

```
<sc>
  <participant name="xs:NCName"/>
</sc>
```

The `participant` element information item defines a participant in the protocol. It has the following properties:

- A **[local name]** of "participant"
- A **[namespace name]** of "urn:ssdl:sc:v1"
- A REQUIRED `name` attribute information item in the **[attributes]** property

3.1.1. name

The **[normalised value]** of the `name` attribute information item is the name of the defined participant. It has the following properties:

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

A `sc` element information item MUST NOT contain two `participant` element information items with the same **[normalised value]** for the `name` attribute information item.

3.2. protocol

```
<ssdl:protocol>
  <sc>
    <protocol name="xs:NCName">
      [ <sequence /> |
        <choice /> |
        <parallel /> |
        <multiple /> |
        <nothing /> |
        <protocolref /> |
        <ssdl:msgref /> ] +
    </protocol>
  </sc>
</ssdl:protocol>
```

The `protocol` element information item defines the structure of a message exchange protocol between `participant`s. It has the following properties:

- A **[local name]** of "protocol"
- A **[namespace name]** of "urn:ssdl:sc:v1"
- A REQUIRED `name` attribute information item in the **[attributes]** property
- One or more of the following element information items in its **[children]** property:
    - A `sequence` element information item
    - A `choice` element information item
    - A `parallel` element information item
    - A `multiple` element information item
    - A `nothing` element information item
    - A `protocolref` element information item
    - A `ssdl:msgref` element information item

3.2.1. name

The **[normalised value]** of the `name` attribute information item is the name of the defined `protocol`. It has the following properties:

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

A `sc` element information item MUST NOT contain two `protocol` element information items with the same **[normalised value]** for the `name` attribute information item.

3.2.2. sequence

```
<ssdl:protocol>
  <sc>
    <protocol>
      <sequence>
      [ <sequence /> |
        <choice /> |
        <parallel /> |
        <multiple /> |
        <nothing /> |
        <protocolref /> |
        <ssdl:msgref /> ] +
      </sequence>
    </protocol>
  </sc>
</ssdl:protocol>
```

The `sequence` element information item defines a sequence of actions, represented by its **[children]** which must occur in sequential order. The sequence element information item has the following properties:

- A **[local name]** of "sequence"
- A **[namespace name]** of "urn:ssdl:sc:v1"
- Two or more of the following element information items in its **[children]** property:
    - A `sequence` element information item
    - A `choice` element information item

- A `parallel` element information item
- A `multiple` element information item
- A `nothing` element information item
- A `protocolref` element information item
- A `ssdl:msgref` element information item

3.2.3. choice

```
<ssdl:protocol>
  <sc>
    <protocol>
      <choice>
      [ <sequence /> |
        <choice /> |
        <parallel /> |
        <multiple /> |
        <nothing /> |
        <protocolref /> |
        <ssdl:msgref /> ] +
      </choice>
    </protocol>
  </sc>
</ssdl:protocol>
```

The `choice` element information item defines an exclusive choice between two or more actions represented by element information items in its **[children]** property. The `sequence` element information has the following properties:

- A **[local name]** of "choice"
- A **[namespace name]** of "urn:ssdl:sc:v1"
- Two or more of the following element information items in its **[children]** property:
  - A `sequence` element information item
  - A `choice` element information item
  - A `parallel` element information item
  - A `multiple` element information item
  - A `nothing` element information item
  - A `protocolref` element information item
  - A `ssdl:msgref` element information item

3.2.4. parallel

```
<ssdl:protocol>
  <sc>
    <protocol>
      <parallel>
        [ <sequence /> |
          <choice /> |
          <parallel /> |
          <multiple /> |
          <nothing /> |
          <protocol /> |
          <ssdl:msgref /> ] +
      </parallel>
    </protocol>
  </sc>
</ssdl:protocol>
```

The `parallel` element information item defines a set of actions, represented by element information items in its **[children]** property, which can occur concurrently. The set of actions

occurring concurrently is only considered complete once all actions have completed. The parallel elopement information has the following properties:

- A **[local name]** of “parallel”
- A **[namespace name]** of “urn:ssdl:sc:v1”
- Two or more of the following element information items in its **[children]** property:
    - A `sequence` element information item
    - A `choice` element information item
    - A `parallel` element information item
    - A `multiple` element information item
    - A `nothing` element information item
    - A `protocol` element information item

3.2.5. multiple

```
<ssdl:protocol>
  <sc>
    <protocol>
      <multiple>
      [ <sequence /> |
        <choice /> |
        <parallel /> |
        <multiple /> |
        <nothing /> |
        <protocol /> |
        <ssdl:msgref /> ] +
      </multiple>
    </protocol>
 </sc>
</ssdl:protocol>
```

The `multiple` element information item defines a set of actions, represented by element information items in its **[children]** property, which can occur a multiple number of times in parallel. One set of actions need not be complete for another to begin. The `multiple` element information item has the following properties:

- A **[local name]** of “multiple”
- A **[namespace name]** of “urn:ssdl:sc:v1”
- Two or more of the following element information items in its **[children]** property:
    - A `sequence` element information item
    - A `choice` element information item
    - A `parallel` element information item
    - A `multiple` element information item
    - A `nothing` element information item
    - A `protocol` element information item

3.2.6. nothing

```
<nothing/>
```

The nothing element information item defines a null action. It has the following properties:

- A **[local name]** of “nothing”
- A **[namespace name]** of “urn:ssdl:sc:v1”

3.2.7. protocolref

```
<protocolref ref="xs:NCName" />
```

The `protocolref` element information item represents an inclusion to a `protocol` defined in the **[children]** property of the `sc` element information item. The `protocolref` element is provided only for syntactic convenience and is equivalent to the element information items present in the **[children]** property of the referenced `protocol` being present before the Sequencing Constraints SSDL Infoset was validated. The `protocolref` element has the following properties:

- A **[local name]** of "protocolref"
- A **[namespace name]** of "urn:ssdl:sc:v1"
- An REQUIRED `ref` attribute information item

3.2.7.1. ref

The `ref` attribute information item must have the same **[normalised value]** as one of the `name` attribute information items present in the **[attribute]** property of the `protocol` element information items. The ref attribute information item has the following properties:

- A **[local name]** of "ref"
- A **[namespace name]** of "urn:ssdl:sc:v1"

3.2.8. ssdl:msgref

```
<ssdl:msgref
  sc:participant="xs:NCName"
  sc:participant-binding-name="xs:NCName" ?
/>
```

An `ssdl:msgref` element information item represents the action of sending or receiving of a message to or from a participant. The action is considered to have occurred 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"

The `ssdl:msgref` element information item MUST have a `participant` attribute information item defined in the Sequencing Constraints SSDL namespace. The `ssdl:msgref` element information item MAY have a `participant-binding-name` attribute information item defined in the Sequencing Constraints SSDL namespace.

3.2.8.1. participant (with ssdl:msgref as [owner element])

The `participant` attribute information item defines the participant who is sending or receiving the message referenced by the **[parent]** element information item. The `patricipant` attribute information item must have the same **[normalised value]** as one of the `name` attribute information items which are in the **[children]** property of the `participant` element information items. The `participant` attribute information item has the following properties:

- A **[local name]** of "participant"
- A **[namespace name]** of "urn:ssdl:sc:v1"

3.2.8.2. participant-binding-name

The `participant-binding-name` attribute information item defines a participant which is to be dynamically bound at runtime to part of the message defined by the **[parent]** element information item. The `participant-binding-name` attribute information item must have the

same **[normalised value]** as one of the `name` attribute information items which are in the **[children]** property of the `participant` element information items.

- A **[local name]** of “participant-binding-name”
- A **[namespace name]** of “urn:ssdl:sc: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] R. Milner, "The Polyadic pi-Calculus: A Tutorial," in *Logic and Algebra of Specification*, F. L. Bauer, W. Brauer, and H. Schwichtenberg, Eds. Berlin, Heidelberg: Springer-Verlag, 1993, pp. 203-246.

[8] S. Woodman, D. Palmer, S. Shrivastava, and S. Wheater, "Notations for the Specification and Verification of Composite Web Services," presented at 8th IEEE International Enterprise Distributed Object Computing (EDOC) Conference, Monterey, California, 2004.

[9] 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.

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

[11] 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: 2 $
-->
<xs:schema
      elementFormDefault="qualified"
      targetNamespace="urn:ssdl:csp:v1"
      xmlns:ssdl="urn:ssdl:v1"
      xmlns:csp="urn:ssdl:csp:v1"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:import namespace="urn:ssdl:v1" />

   <xs:element name="sc">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="participant" type="participant-type"
                        minOccurs="1" maxOccurs="unbounded"/>
            <xs:element name="protocol" type="protocol-type"
                        minOccurs="1" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

```
   <xs:complexType name="participant-type">
      <xs:attribute name="name" type="xs:NCName" use="required"/>
      <xs:attribute name="abstract" type="xs:boolean"
                    use="optional" default="false"/>
   </xs:complexType>

   <xs:complexType name="protocol-type">
      <xs:group ref="sc:action-type" minOccurs="1" maxOccurs="unbounded"/>
      <xs:attribute name="name" type="xs:NCName" use="optional"/>
   </xs:complexType>

   <xs:attribute name="participant" type="xs:string"/>

   <xs:attribute name="participant-binding-name" type="xs:NCName"/>

   <xs:group name="action-type">
      <xs:choice>
         <xs:element name="sequence" type="sc:sequence-type"/>
         <xs:element name="choice" type="sc:choice-type"/>
         <xs:element name="parallel" type="sc:parallel-type"/>
         <xs:element name="multiple" type="sc:multiple-type"/>
         <xs:element name="nothing" type="sc:nothing-type"/>
         <xs:element name="protocolref" type="sc:protocol-ref-type"/>
         <xs:element ref="ssdl:msgref" minOccurs="1" maxOccurs="1"/>
      </xs:choice>
   </xs:group>

   <xs:complexType name="sequence-type">
      <xs:group ref="sc:action-type" minOccurs="2" maxOccurs="unbounded"/>
   </xs:complexType>

   <xs:complexType name="choice-type">
      <xs:group ref="sc:action-type" minOccurs="2" maxOccurs="unbounded"/>
   </xs:complexType>

   <xs:complexType name="parallel-type">
      <xs:group ref="sc:action-type" minOccurs="2" maxOccurs="unbounded"/>
   </xs:complexType>

   <xs:complexType name="multiple-type">
      <xs:group ref="sc:action-type"/>
   </xs:complexType>

   <xs:simpleType name="nothing-type">
      <xs:restriction base="xs:string">
         <xs:enumeration value=""/>
      </xs:restriction>
   </xs:simpleType>

   <xs:complexType name="protocol-ref-type">
      <xs:attribute name="ref" type="xs:QName"/>
   </xs:complexType>
</xs:schema>
```