Simple Events Push for WS-Manageability 1.0

Authors
Igor Sedukhin, Computer Associates (igor.sedukhin@ca.com)
Mark Potts, Talking Blocks (mark.potts@talkingblocks.com)

Copyright
This document is a technical note sent to the open discussion list of the OASIS WSDM TC (wsdm@oasis-open.org). The authors provide this document as input and to aid future discussions within the TC. This document and its contents can be freely distributed and copied provided that reference to the authors and source of the document is included with the copy.

Status and Scope of this Document
The document is provided as a clarification to the submission of the WS-Manageability 1.0 specification to the OASIS Web Services Distributed Management Technical Committee. This document merely explains how to apply existing standards, such as WSDL, to describe a simple push mechanism applicable to event aspects defined within the WS-Manageability 1.0 and their implementation. The authors believe that a general event mechanism for Web services should be standardized outside of the WSDM TC, and this document is not a proposal for general Web services events mechanism.

1 Introduction

The WS-Manageability 1.0 – Specification document presented a Web service endpoint manageability model in terms of topics and aspects. One of the aspects is events. Particular topic model may define particular event information. For example, the state topic defines the StateChangeInformation data type. The information is then contained in an event description along with other common description of the situation (Common Base Event). The topic model may define named events with particular situational semantics. Named events imply under which circumstances, what kinds of information needs to be emitted in an event description. For example the state topic defines the stateChanged named event.

However the model does not imply any particular mechanism for the delivery of event descriptions or the registration of interest in particular events. With the absence of such standard mechanism defined for WS-I 1.0 compliant Web services today, the complimentary WS-Manageability 1.0 – Representation document presented a simple event polling approach defined in a WSDL 1.1 portType (CommonEventsAccess in urn:wsdm:common namespace). The polling mechanism is assumed to be replaced by a more prominent notification mechanism when it becomes available for WS-I compliant Web services (pure Web services, per se).

In certain situations, the event polling mechanism is not sufficient or even applicable at all. With that, it may need to be addressed today for WS-I 1.0 compliant Web services implementations. It may not be sufficient to wait until event push is addressed by some standard specification and adopted by Web services platform vendors and industry in

general. In the mean time, WS-I 1.0 compliant Web services have to be made manageable in a standard way and events push may be required for efficient and working implementations.

The document introduces a clarification to an earlier WS-Manageability 1.0 submission to the OASIS WSDM TC by IBM, CA and Talking Blocks. The clarification is a very simple event push mechanism described using WSDL 1.1, WS-Addressing and CBE and is compliant with WS-I 1.0 Web services and does not require modifications or augmentation of the Web services platforms and tools. The intention of this clarification is to allow efficient and capable implementations of the WS-Manageability 1.0 specification.

This document uses certain constructs defined in the WS-Manageability 1.0 specification.

This document does not try to address event management generally. That could be introduced later by an extension or otherwise externally. This document describes a simple peer-to-peer notification approach applicable to WS-Manageability 1.0 implementations.

2 Simple Events Push

To briefly recap the requirements and definitions:

- Events are named elements of the model.
- Events can be identified. Identifiers include, but not limited to: a property QName, a transition URI, a name string.
- Named events may have specific information defined in the model
- Event information is packaged into a Common Base Event description as an extended data element.
- Need a WSDL 1.1 description of the events push approach: registration of the interest and a deliverable recipient.

There are two WSDL 1.1 portTypes defined in the sections below: one for registration of the interest in the events and another one for a deliverable recipient. These portTypes are intended to be aggregated with other manageability portTypes and then either consolidated (composed) into the functional portType offered by the manageable functional Web service endpoint, or offered by the dedicated manageability endpoint associated with the functional Web Service endpoint (see WS-Manageability 1.0 – Concepts section 5.1.1).

The deliverable recipient portType has to be composed (consolidated) into the Web service endpoint exposed by the consumer of events. For example, a manager could compose deliverable recipient operations and messages into the “manager’s Web service”.

2.1 How does it work

Following are XML message exchange examples that demonstrate what do the event producer and event consumer do to make the simple events push mechanism work.

Assume the following namespace declarations.

```
xmlns:p=”urn:wsdm:common:events:access:push:types”
```

```
xmlns:c=”urn:wsdm:common:events:recipient:types”
xmlns:cmn="urn:wsdm:common"
xmlns:cbe="http://www.ibm.com/AC/commonbaseevent1_1"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
```

To register an interest, the event consumer sends the following XML in a SOAP envelope to the event producer.

```
<p:RegisterInterestInEvents>
<p:eventIndentifiers>
      <cmn:eventIdentifier>
      <cmn:transition>urn:wsdm:webservice:endpoint:lifecycle:up:TO:down
      </cmn:transition>
      <cmn:eventIdentifier>
</p:eventIdentifiers>
<p:recipientEndpoint>
      <wsa:Address>http://myRecipient.mycorp.com/listener</wsa:Address>
</p:recipientEndpoint>
</p:RegisterInterestInEvents>
```

The response to the above request is the following XML in a SOAP envelope or any SOAP fault.

```
<p:RegisterInterestInEventsResponse>
<p:registration> […some XML…] </p:registration>
</p:RegisterInterestInEventsResponse>
```

To verify that registration is in place, the event consumer sends the following XML in a SOAP envelope to the event producer.

```
<p:VerifyInterestInEvents>
<p:registration> […some XML…] </p:registration>
</p:VerifyInterestInEvents>
```

The response to the above request is the following XML in a SOAP envelope or any SOAP fault.

```
<p:VerifyInterestInEventsResponse>
<p:eventIndentifiers>
      <cmn:eventIdentifier>
      <cmn:transition>urn:wsdm:webservice:endpoint:lifecycle:up:TO:down
      </cmn:transition>
      <cmn:eventIdentifier>
</p:eventIdentifiers>
</p:VerifyInterestInEventsResponse>
```

To cancel the interest, the event consumer sends the following XML in a SOAP envelope to the event producer.

```
<p:CancelInterestInEvents>
<p:registration> […some XML…] </p:registration>
</p:CancelInterestInEvents>
```

To cancel the interest in particular events, the event consumer sends the following XML in a SOAP envelope to the event producer.

```
<p:CancelInterestInEvents>
<p:registration> […some XML…] </p:registration>
<p:eventIndentifiers>
      <cmn:eventIdentifier>
      <cmn:transition>urn:wsdm:webservice:endpoint:lifecycle:up:TO:down
      </cmn:transition>
      <cmn:eventIdentifier>
</p:eventIdentifiers>
</p:CancelInterestInEvents>
```

The response to either of the above requests is the following XML in a SOAP envelope or any SOAP fault.

```
<p:CancelInterestInEventsResponse>
<p:OK>true</p:OK>
</p:CancelInterestInEventsResponse>
```

To receive an event notification, the event consumer implements the EventsPushRecipient portType. That means the event consumer must be capable to accept the following XML in a SOAP envelope.

```
<c:ReceiveEvent>
<c:registration> […some XML…] </c:registration>
<c:eventIdentifier>
      <cmn:transition>urn:wsdm:webservice:endpoint:lifecycle:up:TO:down
      </cmn:transition>
<c:eventIdentifier>
<c:eventDescription>
[…CBE data…]
</c:eventDescription>
</c:ReceiveEvent>
```

The event consumer should respond with the following XML is a SOAP envelope or any SOAP fault.

```
<c:ReceiveEventResponse>
<c:OK>true</c:OK>
</c:ReceiveEventResponse>
```

To understand that registration was cancelled by the producer, the events consumer needs to process the following XML in a SOAP envelope. Essentially scan XML for the RegistrationCancelled element QName.

```
<c:ReceiveEvent>
<c:registration> […some XML…] </c:registration>
<c:eventIdentifier>
      <cmn:transition>urn:wsdm:webservice:endpoint:lifecycle:up:TO:down
      </cmn:transition>
```

```
<c:eventIdentifier>
<c:eventDescription>
[…CBE data…]
<cbe:extendedDataElement>
<p:RegistrationCancelled> … </p:RegistrationCancelled>
</cbe:extendedDataElement>
[…CBE data…]
</c:eventDescription>
</c:ReceiveEvent>
```

2.2 Description of the Registration of the Interest

```
<definitions name=”EventsPushAccess”
 targetNamespace=”urn:wsdm:common:events:access:push”
 xmlns:tns=”urn:wsdm:common:events:access:push”
 xmlns:tns-typ=”urn:wsdm:common:events:access:push:types”
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns=”http://schemas.xmlsoap.org/wsdl/”>

<types>
<xsd:schema
 targetNamespace="urn:wsdm:common:events:access:push:types"
 xmlns:cmn="urn:wsdm:common"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
 elementFormDefault="qualified">

<xsd:import
 namespace="urn:wsdm:common"
 schemaLocation="../../schema/Common.xsd"/>

<xsd:import
 namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
 schemaLocation="../../schema/wsa.xsd"/>

<xsd:element name=”RegisterInterestInEvents”>
      <xsd:sequence>
            <xsd:element name=”eventIdentifiers”
                  type=”cmn:ArrayOfEventIdentifiers”/>
            <!-- the recipientEndpoint must be able to accept
                 messages as defined in the EventsPushRecipient
                 portType in the urn:wsdm:common:events:recipient
                 namespace -->
            <xsd:element name=”recipientEndpoint”
                  type=”wsa:EndpointReferenceType”/>
      </xsd:sequence>
</xsd:element>

<xsd:element name=”RegisterInterestInEventsResponse”>
      <xsd:sequence>
            <xsd:element name=”registration” type=”xsd:anyType”/>
      </xsd:sequence>
</xsd:element>

<xsd:element name=”CancelInterestInEvents”>
      <xsd:sequence>
```

```
            <xsd:element name=”registration” type=”xsd:anyType”/>
            <xsd:element name=”eventIdentifiers”
                  type=”cmn:ArrayOfEventIdentifiers”
                  minOccurs=”0”/>
      </xsd:sequence>
</xsd:element>

<xsd:element name=”CancelInterestInEventsResponse”>
      <xsd:sequence>
            <xsd:element name=”OK” type=”xsd:boolean”/>
      </xsd:sequence>
</xsd:element>

<xsd:element name=”VerifyInterestInEvents”>
      <xsd:sequence>
            <xsd:element name=”registration” type=”xsd:anyType”/>
      </xsd:sequence>
</xsd:element>

<xsd:element name=”VerifyInterestInEventsResponse”>
      <xsd:sequence>
            <xsd:element name=”eventIdentifiers”
                  type=”cmn:ArrayOfEventIdentifiers”/>
      </xsd:sequence>
</xsd:element>

<!-- this element appears in the CBE extendedDataElement
     it indicates that a resource will not be able to meet
     the specified registration for the interest in events
     anymore -->
<xsd:element name=”RegistrationCancelled”>
      <xsd:sequence>
            <xsd:element name=”registration” type=”xsd:anyType”/>
      </xsd:sequence>
</xsd:element>

</xsd:schema>
</types>

<message name="RegisterInterestInEventsIn">
      <part name="parameters"
            element="tns-typ:RegisterInterestInEvents"/>
</message>

<message name="RegisterInterestInEventsOut">
      <part name="parameters"
            element="tns-typ:RegisterInterestInEventsResponse"/>
</message>

<message name="CancelInterestInEventsIn">
      <part name="parameters"
            element="tns-typ:CancelInterestInEvents"/>
</message>

<message name="CancelInterestInEventsOut">
      <part name="parameters"
            element="tns-typ:CancelInterestInEventsResponse"/>
```

```
</message>

<message name="VerifyInterestInEventsIn">
      <part name="parameters"
            element="tns-typ:VerifyInterestInEvents"/>
</message>

<message name="VerifyInterestInEventsOut">
      <part name="parameters"
            element="tns-typ:VerifyInterestInEventsResponse"/>
</message>

<portType name="EventsPushAccess">

      <operation name="RegisterInterestInEvents">
            <input message="tns:RegisterInterestInEventsIn"/>
            <output message="tns:RegisterInterestInEventsOut"/>
      </operation>

      <operation name="CancelInterestInEvents">
            <input message="tns:CancelInterestInEventsIn"/>
            <output message="tns:CancelInterestInEventsOut"/>
      </operation>

      <operation name="VerifyInterestInEvents">
            <input message="tns:VerifyInterestInEventsIn"/>
            <output message="tns:VerifyInterestInEventsOut"/>
      </operation>

</portType>

</definitions>
```

2.3 Description of the Deliverable Recipient

```
<definitions name=”EventsPushRecipient”
 targetNamespace=”urn:wsdm:common:events:recipient”
 xmlns:tns=”urn:wsdm:common:events:recipient”
 xmlns:tns-typ=”urn:wsdm:common:events:recipient:types”
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns=”http://schemas.xmlsoap.org/wsdl/”>

<types>
<xsd:schema
 targetNamespace="urn:wsdm:common:events:recipient:types"
 xmlns:cmn="urn:wsdm:common"
 xmlns:cbe="http://www.ibm.com/AC/commonbaseevent1_1"
 elementFormDefault="qualified">

<xsd:import
 namespace="urn:wsdm:common"
 schemaLocation="../../schema/Common.xsd"/>

<xsd:import
 namespace="http://www.ibm.com/AC/commonbaseevent1_1"
```

```
 schemaLocation="cbe.xsd"/>

<xsd:element name=”ReceiveEvent”>
      <xsd:sequence>
            <xsd:element name=”registration”
                  type=”xsd:anyType”/>
            <xsd:element name=”eventIdentifier”
                  type=”cmn:EventIdentifier” minOccurs=”0”/>
            <xsd:element name=”eventDescription”
                  type=”cbe:CommonBaseEventType”/>
      </xsd:sequence>
</xsd:element>

<xsd:element name=”ReceiveEventResponse”>
      <xsd:sequence>
            <xsd:element name=”OK” type=”xsd:boolean”/>
      </xsd:sequence>
</xsd:element>

</xsd:schema>
</types>

<message name="ReceiveEventIn">
      <part name="parameters" element="tns-typ:ReceiveEvent"/>
</message>

<message name="ReceiveEventOut">
      <part name="parameters" element="tns-typ:ReceiveEventResponse"/>
</message>

<portType name="EventsPushRecipient">

      <operation name="ReceiveEvent">
            <input message="tns:ReceiveEventIn"/>
            <output message="tns:ReceiveEventOut"/>
      </operation>

</portType>

</definitions>
```