Satisfaction of Requirements of 6401-6661

Li Li, Wu Chou 07/20/09

I. Event Sink Code Generation

It must be possible for a developer of an Event Sink to generate code that dispatches and marshals Notifications using current WSDL-based tools. There are three sub-cases:

A. Raw Notifications

It must be possible for developers to do the above for Raw Notifications.

B. Wrapped notifications

It must be possible for developers to do the above for Wrapped Notifications.

C. Extension Notifications

It should be possible for developers to do the above for Notifications who's OTW shape has been changed by an extension to WS-Eventing (e.g. a new Format). Since the WG cannot know at this time what sorts of extensions may be invented, it is impossible to determine whether any solution can satisfy this requirement for all possible extensions. At the very least, no solution for this requirement should do anything to make it impossible to support extensions that change the shape of the OTW Notification. A non-functional requirement of this use case is that it should be "easy" to support the kinds of extensions that are currently envisioned (e.g. a "batched" format).

Satisfaction: Yes. This approach is solely based on WSDL and WS-Policy framework and attachment. For tools that do not support WS-Policy, the policy expressions will be ignored and the tools will not break. Any extension that creates a different OTW shape can and should be represented by WSDL and composed with other formats.

II. Event Type Visibility

A Subscriber can view metadata about the set of potential Events Types (including schemas) that may be emitted by an Event Source. A non-functional requirement is that it must be possible to screen this metadata based on authorization decisions about the identity of the Subscriber.

Satisfaction: Yes. The event types can be represented as one-way operations in the notification WSDL file, which is grouped into port types. Because the event types are in the notification WSDL which is separated from but linked by the event source WSDL, authorization can be applied easily. For example, when the Subscriber tries to retrieve the WSDL from the links, credentials must be supplied.

III. Event Type Completeness

It must be possible for a service designer to take the metadata about the set of potential Event Types that may be emitted by an Event Source and implement the Event Sink.

Satisfaction: Yes. The event types can be represented as one-way operations in the notification WSDL file, which is grouped into port types. To implement the sink, a developer just need to select the bindings for the port types, run the tools and write code.

IV. Non-Metadata Use Cases

It must be possible to realize ALL USECASES without the use of metadata.

Satisfaction: Yes. ALL USECASES can be realized without the use of metadata.

V. Multiple Notification Variations

It must be possible for a single Event Source to transmit multiple variations of Notifications (Raw, Wrapped, *) for the same Event Type.

Satisfaction: Yes. Notification variations are specified using WSDLs. Each variation can be specified in the same or different WSDL files.

VI. Advertise Notification Variations

It must be possible to provide a Subscriber with metadata that describes the variations of Notifications (e.g. supported formats)

Satisfaction: Yes. Notification variations are specified using WSDLs. Each variation can be specified in the same or different WSDL files. These WSDL files are used by Subscribers or tools to select desired notifications.

VII. Deterministic WSDL

It must be possible for a Subscriber to determine the WSDL that describes the interface that the Event Sink needs to implement based on the various parameters and extensions in the Subscribe request.

Satisfaction: Yes. Since all parts of the WSDL are linked and event source WSDL files and notification WSDL files are also linked by policies, a Subscriber can follow the links to determine the desired interfaces.

VIII. WS-I Basic Profile

WSDLs of Event Source and Event Sink must be WS-I Basic Profile compliant.

Satisfaction: Yes. Neither event source nor notification WSDL contains any outbound operations.

IX. Extensibility of Message Exchange Patterns (MEPs)

Although WS-Eventing is only required to support simple one-way Notifications, a solution should not prohibit the extension of message exchange patterns to support a variety of event notifications that can fit into WSDL 1.1 and WSDL 2.0 framework, e.g. WS-Management specifies that a notification can be acknowledged, and ECMA CSTA requires that the Event Sink responds to the notification with certain type of messages.

Satisfaction: Yes. Since this approach uses WSDLs, it allows service designers to define any MEP in the notification WSDL that is necessary for the application.

Global Non-Functional Requirements

Any solution to the above use cases should satisfy the following non-functional requirements:

Constrained Environments

All use cases must have 'reasonable' solution for constrained environments.

Satisfaction: Yes. Since this approach relies only on WSDLs and policy is optional, it does not require any extra processing other than generating code from WSDL. Even this step can be performed offline. It supports runtime or design time processing while tools can read WSDL and generate code stubs.

Known Technologies

Any solution to the above use cases must limit inventions to applications of well known technologies (e.g. WSDL, WS-Policy, WS-MetadataExchange).

Satisfaction: Yes. Since this approach relies only on WSDL and WS-Policy, which is optional.

Consistent with UDDI

Although WS-Eventing is not required to provide explicit support for UDDI, a solution should not prohibit either Event Sources or Event Sinks from publishing and registering

their services through UDDI, so that they can be discovered and consumed through the existing web service infrastructure.

Satisfaction: Yes. WSDL can be decomposed and published in UDDI registry, and this process does not effect the policy attachments. For details, please see a tutorial at http://www.ibm.com/developerworks/webservices/library/ws-wsdl/.