WS-I Usage Scenarios

Document Status: WS-I Board Approval Draft

Version: 1.01

Date: April 14, 2003

Editors: Scott Werden, WRQ
Colleen Evans, Sonic Software
Marc Goodner, SAP

Table of Contents

Table of Figures

Revision History

Version 1.0	Original version
Version 1.01	Revisions to Board Approval Draft to reflect late changes to the Basic Profile 1.0

Copyright

Confidentiality

This document contains proprietary information that is confidential and shall not be made available to unauthorized persons.

1 Introduction

WS-I Usage Scenarios define the use of Web services in structured interactions, identifying basic interoperability requirements for such interactions and mapping the flow of a scenario to the requirements of the WS-I Basic Profile 1.0 (hereafter, Basic Profile) [1]. Scenarios are independent of any application domain. WS-I Use Cases employ Scenarios to model high-level definitions of specific applications.

The scenarios presented here can be composed or extended. That is, they describe fundamental Web service design patterns that can be combined and built upon like building blocks. For example, the Synchronous Request/Response scenario describes a basic exchange and can be expanded by adding SOAP headers. The only requirement is that the extensions must also conform to the Basic Profile.

1.1 How to use this document

This document describes the WS-I Usage Scenarios to be used with the Basic Profile. The Basic Profile constraints and requirements are referenced directly and the reader is expected to use the Basic Profile in conjunction with this document to interpret the referenced information.

The three scenarios presented in this document are intended to provide sufficient information so that a user of this document can create WS-I compliant Web service applications using one of more of the scenarios. All applicable guidelines and restrictions for the messages and service description instances for each scenario are provided.

2 Usage Scenario Taxonomy

The Usage Scenario taxonomy defines a structure for applying the Basic Profile constraints. The taxonomy consists of a Web services stack and a set of activities, grouped by the layers of the stack, that a Web service instance executes as part of the Web service Usage Scenario. The constraints of the Basic Profile are applied to each activity as well as to the optional components of the scenario, e.g., the WSDL for the description of the Web service instance. There are two types of constraints on scenarios:

- Flow Constraints applying to each activity that takes part in the flow of the Web service. These include: expressing the Web service data model in XML, creating and consuming messages using SOAP, transporting messages using HTTP
- Description Constraints applying to the *description* of the Scenario. Operationally, the description of a Web service instance occurs in WSDL and possibly UDDI, therefore, these constraints are applied to the WSDL and UDDI describing the Scenario.

The following are attributes of WS-I Usage Scenarios:

- They include a flow description, linking together the set of activities specific to the scenario,
- They include optional components, such as SOAP headers or security,
- They are described with a WSDL document,
- Each activity within a scenario has constraints applied to it by the Basic Profile, and
- They represent a real-world Web service implementation.

2.1 Web Service Stack

The Usage Scenario taxonomy is based on a Web services stack. Each layer of the stack represents one of the fundamental functional areas of a Web service instance. Not all possible functional areas are represented (e.g., security or coordination), only the most basic. These layers are depicted in the following diagram.

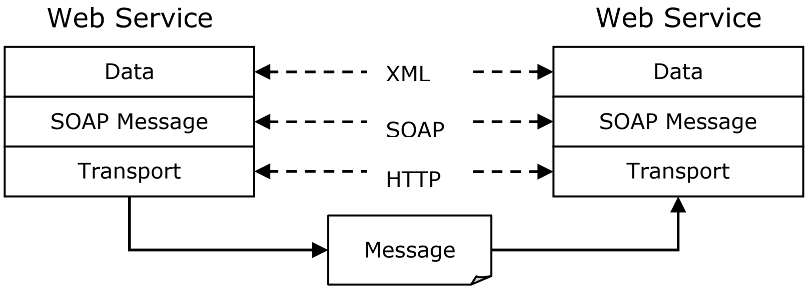

Figure 2-1 Web services stack

A Web service application may include several logical layers incorporating functions such as the Web service instance and application business logic. The Basic Profile and Usage Scenarios do not address application business logic except where the functionality of any part of the Web services stack is implemented within the business logic.

The details of each layer of the Web service stack are:

Data Layer

The data layer translates the application specific data into the model chosen for the specific Web service. The data layer includes the functions necessary to support flexible data typing. This layer maps to the `wsdl:types` and `wsdl:message` definitions within a WSDL document.

SOAP Message Layer

The SOAP message layer is the infrastructure that processes SOAP messages, dispatches them, and may optionally fulfill Quality of Service requirements. On the sending side the message layer writes SOAP messages, based on the data model defined in portTypes and bindings. On the receiving side the message layer processes the SOAP messages and dispatches requests to the correct application or method.

Transport Layer

The transport layer sends and receives messages. For the Basic Profile, this includes only HTTP client and server platforms. This layer maps to the `wsdl:binding` and `wsdl:port` definitions with the WSDL document.

2.2 Activities

A set of activities is defined for each layer of the Web service stack. Activities are the fundamental operations that comprise a Web service. A single activity has several constraints applied to it from the Basic Profile. For example, one activity might be “Send HTTP” and the specifications and guidelines for how to fulfill that activity come from the SOAP 1.1 and HTTP sections of the Basic Profile.

The following table summarizes these activities.

Layer	**Activity**
Data Layer	Write XML

Layer	Activity
	Process XML
SOAP Message Layer	Write SOAP envelope Process SOAP envelope Write SOAP body Process SOAP body Write SOAP header Process SOAP header
Transport Layer	Send HTTP Receive HTTP

Table 1 - Activities grouped by Web services stack layer

2.2.1 Data Layer Activities

The following activities are part of the Data layer:

Write XML

Application-level messages that are to be exchanged during a Web services interaction must be written to a serialized form that can be transported with the underlying transport protocol. These messages use the data types and formats declared in the data model documentation (i.e., WSDL or Schema). Writing the message data is the responsibility of the application component sending a message to a recipient.

Process XML

Application-level messages that are exchanged in a Web services interaction are passed to application components responsible for receiving, interpreting and acting upon the received messages. Application components process message data according to the types and formats declared in the data model documentation.

2.2.2 SOAP Message Layer Activities

The following activities are executed with the SOAP Message Layer.

SOAP envelope

The SOAP envelope is the container for all the other SOAP message parts, including the payload.

- Write SOAP envelope
- Process SOAP envelope

SOAP body

The SOAP body is used for transporting application-specific information included in the application message data. The activities in this layer are different from the data payload writing and processing activities described in the Data Layer activities section.

- Write SOAP body
- Process SOAP body

SOAP header

The SOAP header provides a modular mechanism for extending a SOAP message.

- Write SOAP header
- Process SOAP header

2.2.3 Transport Layer Activities

SOAP messages may be sent using the HTTP or HTTPS transport protocols.

- Send HTTP
- Receive HTTP

2.2.4 Web Service Actors

In WS-I Web services scenarios there are two high level actors. These are not related to SOAP Actors as defined in SOAP 1.1.

Consumer

A Consumer is responsible for making requests of a service implemented by a Provider.

Provider

A Provider is responsible for listening for and processing Consumer service requests.

2.3 Security

Usage scenarios do not explicitly address authentication, authorization, identification, or privacy. However, some of those concerns can be addressed with existing technologies that are compatible with the Basic Profile. For example, the HTTPS binding can be used rather than the un-encrypted HTTP binding. Application level security can always be added within the message layer and this would be entirely transparent to the Basic Profile.

Countermeasures are best applied through a risk assessment of your Web service application. To assist in this process please see the Security Appendix below for more detailed information on common threats and Basic Profile compliant mitigation strategies. Each Usage Scenario includes a section detailing additional concerns as well as the identified common threats most relevant to the given scenario.

3 Usage Scenarios

This section defines the three Usage Scenarios developed to complement the Basic Profile:

- One-way
- Synchronous request/response
- Basic callback

3.1 One-way

3.1.1 Description

A Consumer sends a request message to a Provider.   The Provider receives the message and processes it.

The exchange is one way; no SOAP response message from the Provider is generated or expected. The underlying transport is not required to guarantee delivery of the message to the Provider. Regardless of the protocol implemented by the transport layer, the Consumer receives no acknowledgement above the transport layer that the message was successfully sent, delivered to the intended destination, or received by the Provider.

This Scenario applies to situations where information loss is inconsequential (for example, in a status monitoring scenario where periodic status update events are provided such that if one update event is lost, a subsequent update event will convey correct status).

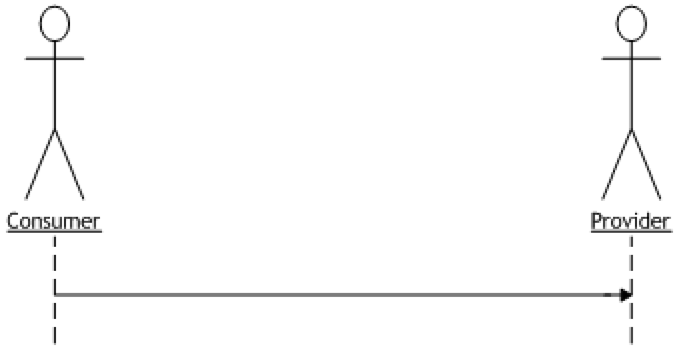

Figure 3-1 One-way Sequence

High-level flow:

1. Consumer invokes the service by sending a SOAP message bound to an HTTP request to the Provider
2. Provider executes the service.

Assumptions:

- This scenario describes a runtime sequence of events; it does not describe the design or deployment activities.
- The data model, the application semantics, and the transport bindings are all agreed upon and implemented a priori to this scenario.
- All parts of this scenario are defined in conformance with the guidelines and recommendations of the Basic Profile.
- This scenario is “composable”, that is, it may be used as a foundation for creating more complex scenarios.

3.1.2 Flow

The detailed flow for this scenario, using the activities defined in Section 2.2, is described below. Each bulleted item represents the activities performed within one layer of the stack required to complete the flow. The order of activities within a Consumer or Provider is not significant. Each activity has constraints imposed upon it from the Basic Profile.

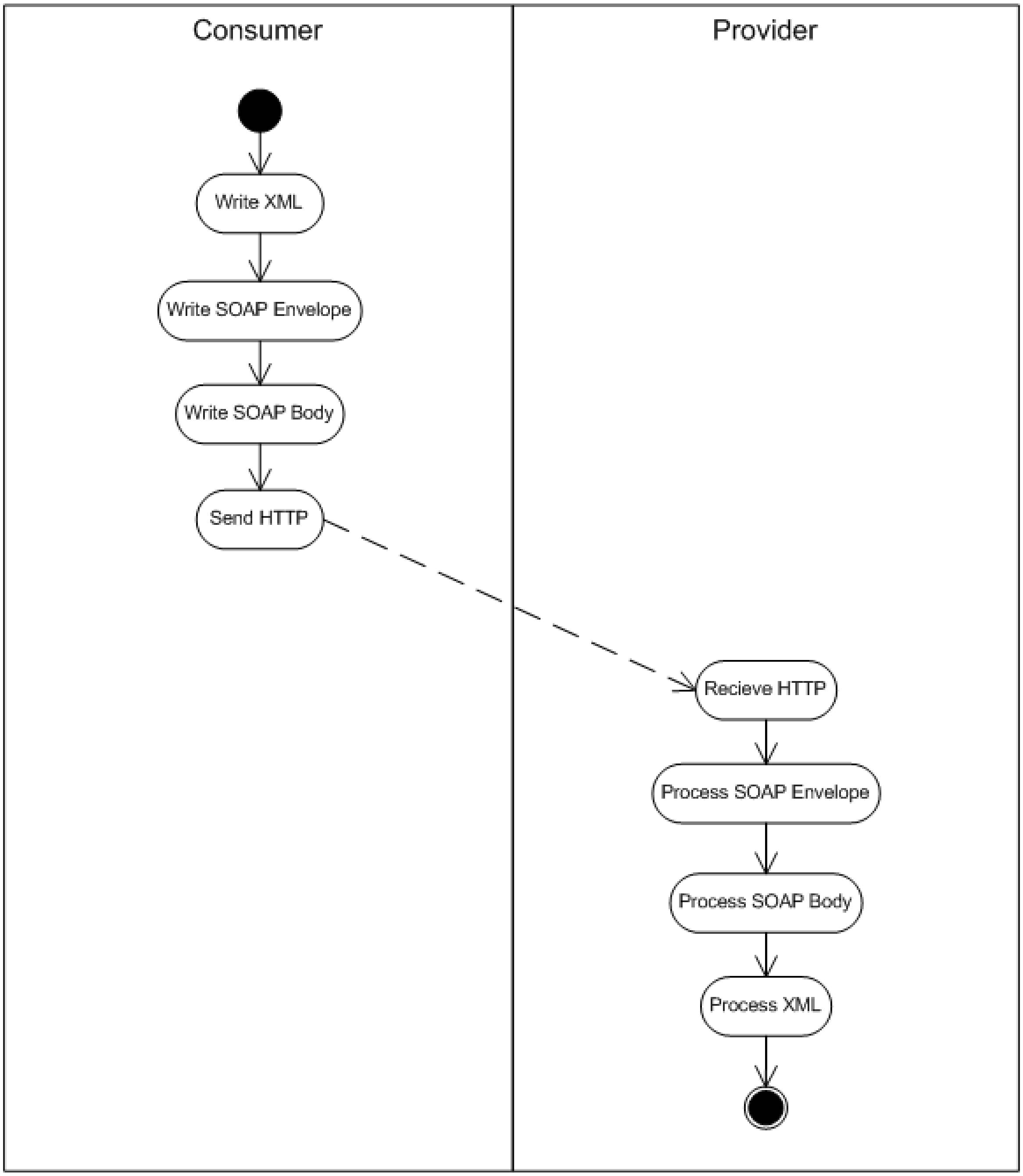

Figure 3-2 One-way Request

The Consumer initiates a SOAP request:

- Data Layer
    - Write XML. The payload is created according to the data model.
- SOAP Message Layer
    - Write SOAP envelope
    - Write SOAP body
- Transport Layer

  - Send HTTP

The Provider receives the SOAP request:

- Transport Layer
  - Receive HTTP
- SOAP Message Layer
  - Process SOAP envelope
  - Process SOAP body
- Data Layer
  - Process XML. The data payload is processed according to the data model and dispatched to the application

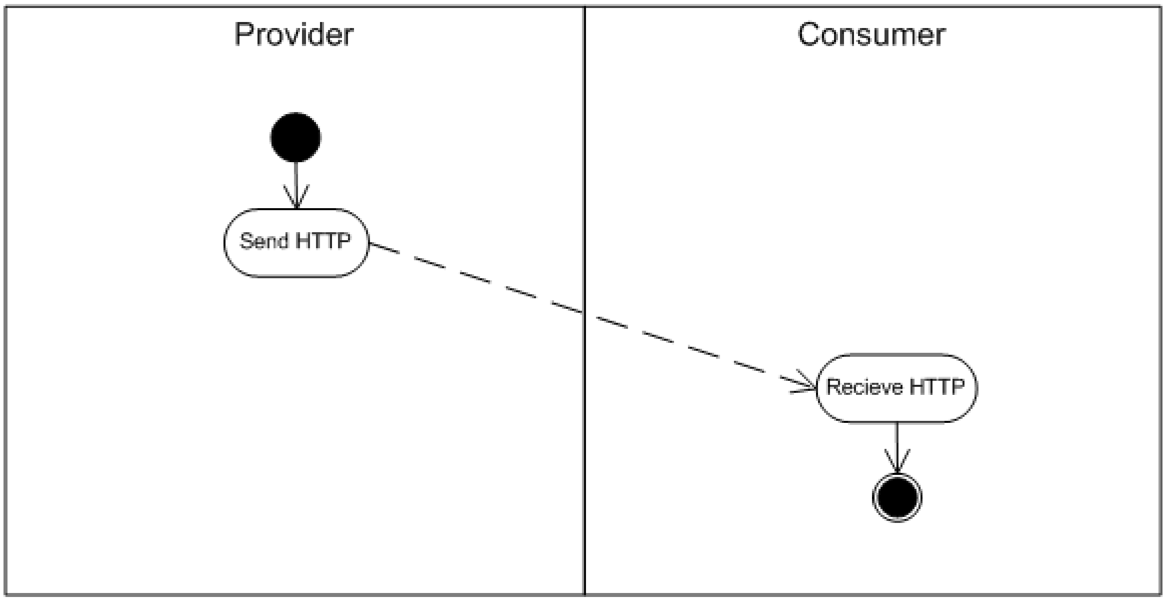

Figure 3-3 One-way Acknowledgement

The Provider sends the acknowledgement:

- Transport Layer
  - Send HTTP. Note that the HTTP response can be sent at any time relative to the processing of the SOAP message. There is no SOAP envelope sent with the HTTP response.

The Consumer receives the acknowledgement:

- Transport Layer
  - Receive HTTP (status code). This is ignored by the higher layers of the Web services stack.

3.1.3 Flow Constraints

The following are the flow constraints upon this Usage Scenario.

- Write XML, as defined in Section 5.1
- Write SOAP envelope, as define in Section 5.3

- Write SOAP body, as define in Section 5.5
- Send HTTP, as defined in Section 5.9. Further constraint specific to this scenario is R2714.
- Receive HTTP, as defined in Section 5.10. Further constraints specific to this scenario are R2727 and R2750.
- Process SOAP envelope, as defined in Section 5.4
- Process SOAP body, as defined in Section 5.6
- Process XML, as defined in Section 5.2

3.1.3.1 Error conditions and SOAP Fault

A SOAP Fault cannot be generated in this scenario since there is no SOAP response message. If any error occurs in the Provider, the Provider and Consumer must respectively abide by R2714.

3.1.3.2 SOAP Headers

Use of a SOAP header is optional for this scenario. If it is used, it must follow the constraints for the Write SOAP Header and Process SOAP Header activities, as defined in Section 5.7 and 5.8, respectively.

3.1.4 Description Constraints

The WSDL should have at least the following content within its definitions for the One-way Scenario. Not all sections are required, but if present in the WSDL, each should follow the guidelines as presented below. General constraints on the WSDL are described in Section 5.11. Other constraints imposed upon the WSDL by the Basic Profile are listed below.

3.1.4.1 types

This section is not required and if present, will be dependent upon the specifics of the data model.

Constraints on WSDL types are listed in Section 5.12.

3.1.4.2 messages

Message format will be dependent upon the data model (doc/literal or rpc/literal). Only one message is defined: one input.

3.1.4.2.1Document messages

Document messages parts are composed from Schema element definitions (see R2204)

```
<wsdl:message …>
        <wsdl:part name=”Input” element=”..”>
</wsdl:message>
```

3.1.4.2.2RPC messages

RPC messages parts are composed from Schema type declarations (see R2203)

```
<wsdl:message …>
        <wsdl:part name=”Input“ type=”..” />
</wsdl:message>
```

Other constraints are listed in Section 5.13.

3.1.4.3 portTypes

The one-way transmission primitive must be used. Grammar is:

```
<wsdl:portType ..>
        <wsdl:operation …>
                <wsdl:input …/>
        </wsdl:operation>
</wsdl:portType>
```

Other constraints are listed in Section 5.14.

3.1.4.4 binding

The `wsdl:binding` section must use the SOAP binding extension with HTTP transport. The same operation type defined in `wsdl:portType` must be used in the binding section.

```
<wsdl:binding …>
        <soap:binding style="rpc|document" transport=http://schemas.xmlsoap.org/soap/http>
                <wsdl:input …>
                </wsdl:input>
        </soap:binding>
</wsdl:binding>
```

Other constraints are listed in Section 5.15.

3.1.4.5 port

The `soap:address` element must be specified along with the URL for the endpoint:

```
<wsdl:port>
        <soap:address location="uri" />
</wsdl:port>
```

Other constraints are listed in Section 5.16.

3.1.5 UDDI

Advertisement of Web services patterned after this scenario adheres to the "Using WSDL in a UDDI Registry, Version 1.07" Best Practice document. A `uddi:tModel` representing the Web service type references the file containing the `wsdl:binding` for the message operation. The `uddi:bindingTemplate` captures the service endpoint and references the `uddi:tModel`(s) for the Web service type.

Advertising Web services in this way enables discovery using the inquiry patterns supported by the UDDI Inquiry API set (see http://www.uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf). These include the browse pattern, the drill-down pattern and the invocation pattern.

General UDDI Constraints are listed in Section 5.17.

3.1.6 Security

This section identifies the threats most relevant to this Usage Scenario as described in Appendix 1 where additional information on Basic Profile compliant countermeasures may also be found.

Additional constraints may apply if HTTPS is used to implement security. These are Profile requirements: R5000, R5001, R5010, . Appendix 1 has additional guidelines on Security.

As of this writing no specific threat has been identified as being singularly relevant to this Usage Scenario.

3.2 Synchronous Request/Response

3.2.1 Description

A Consumer sends a request message to a Provider. The Provider receives the message, processes it, and sends back a response.

The following diagram shows the high-level interactions between a Consumer and a Provider in the Synchronous Request/Response Usage Scenario.

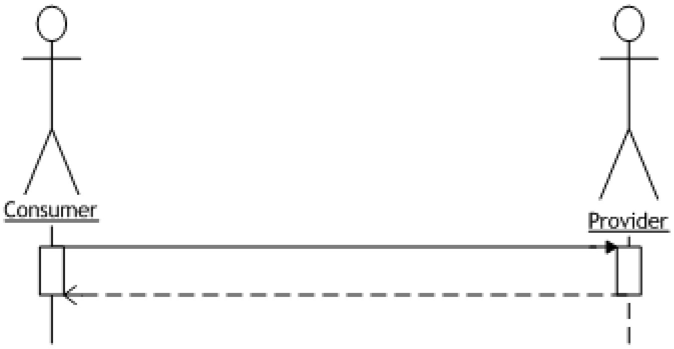

Figure 3-4 Synchronous Request/Response Sequence

High-level flow:

1. Consumer invokes the service by sending a SOAP message bound to an HTTP request to the Provider
2. Provider executes the service and sends a SOAP message bound to an HTTP response to the Consumer

Assumptions:

1. This scenario is a runtime sequence of events; it does not involve any design or deployment activities.
2. The data model, the application semantics, and the transport bindings are all agreed upon and implemented a priori to this scenario.
3. This scenario is "composable", that is, it may be used as a foundation for creating more complex scenarios.
4. The Request and Response messages are synchronized through the HTTP transport.

3.2.2 Flow

The detailed flow for this scenario, using the activities defined in Section 2.2, is described below. Each bulleted item represents the activities performed within one layer of the stack required to complete the flow. The order of activities within a Consumer or Provider is not significant. Each activity has constraints imposed upon it from the Basic Profile.

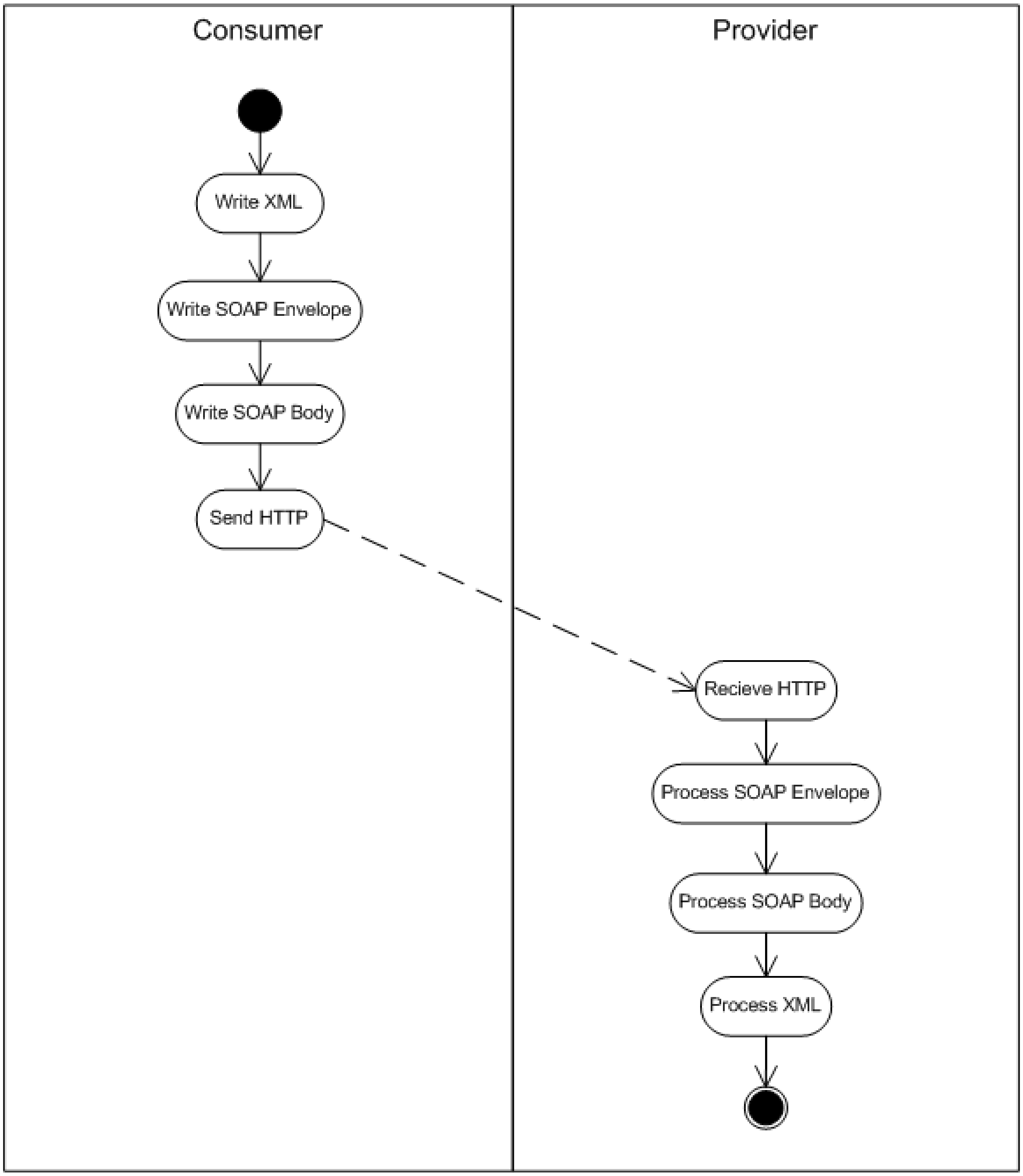

Figure 3-5 Synchronous Request

The Consumer initiates a SOAP request:

- Data Layer
  - Write XML. The payload is created according to the data model.
- SOAP Message Layer

  - Write SOAP envelope
  - Write SOAP body
- Transport Layer
  - Send HTTP

The Provider receives the SOAP request:

- Transport Layer
  - Receive HTTP
- SOAP Message Layer
  - Process SOAP envelope
  - Process SOAP body
- Data Layer
  - Process XML. The data payload is processed according to the data model and dispatched to the application

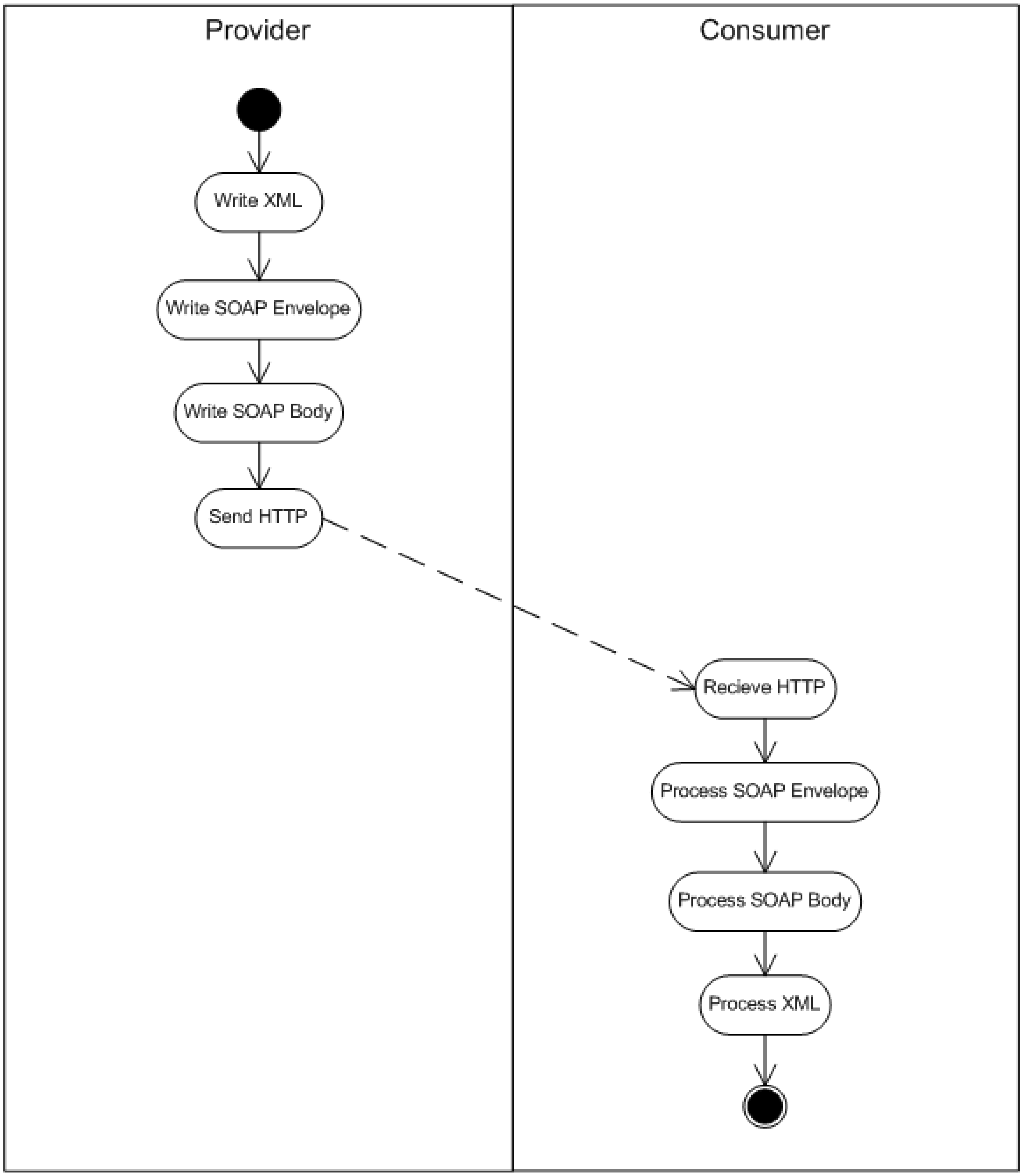

Figure 3-6 Synchronous Response

The Provider generates a SOAP response:

- Data Layer
    - Write XML. The payload is created according to the data model.
- SOAP Message Layer
    - Write SOAP envelope
    - Write SOAP body
- Transport Layer

    - Send HTTP

The Consumer receives the SOAP response:

- Transport Layer
    - Receive HTTP
- SOAP Message Layer
    - Process SOAP envelope
    - Process SOAP body
- Data Layer
    - Process XML. The data payload is processed according to the data model and dispatched to the application

3.2.3 Flow Constraints

The following activities have the referenced constraints in this Usage Scenario.

- Write XML, as defined in Section 5.1
- Write SOAP envelope, as defined in section 5.3
- Write SOAP body, as define in Section 5.5
- Send HTTP, as defined in Section 5.9
- Receive HTTP, as defined in Section 5.10
- Process SOAP envelope, as defined in Section 5.4
- Process SOAP body, as defined in Section 5.6
- Process XML, as defined in Section 5.2

3.2.3.1 Errors and SOAP Faults

Errors that occur during SOAP processing are communicated with a SOAP Fault message, as per the SOAP 1.1 specification. This scenario supports SOAP Faults through composition, that is, all the constraints described in sections 3.2.3 and 3.3.4.1 apply, plus the additional constraints imposed upon the following Activities.

The Web service Provider must abide by the following restrictions and guidelines from the Basic Profile:

- Behavior for fault generation: R2724, R2725
- Writing the `soap:fault`: R1000, R1001, R1004, R1031, R2742, R2743
- HTTP SOAPAction: R1119
- HTTP status codes: R1126
- Requirements for WSDL description: R2728, R2742, R2743, R2754

The Web service Consumer must follow the following restrictions and guidelines from the Basic Profile:

- Processing the soap:fault: R1002, R1003, R1016

3.2.3.2 SOAP Headers

Use of a SOAP header is optional for this scenario. If it is used, it must follow the constraints for the Write SOAP Header and Process SOAP Header activities, as defined in Sections 5.7 and 5.8, respectively.

3.2.4 Description Constraints: WSDL

The WSDL should have at least the following content within its definitions for the Synchronous Request/Response Scenario. Not all sections are required, but if present in the WSDL, each should follow the guidelines as presented below. General constraints on the WSDL are described in Section 5.11. Other constraints imposed upon the WSDL by the Basic Profile are listed below.

3.2.4.1 types

This WSDL section is not required, and if present, will be dependent upon the specifics of the data model.

Constraints on types are listed in Section 5.12.

3.2.4.2 messages

Message format will be dependent upon the data model (doc/literal or rpc/literal). At least two messages must be defined: one input and one output. Optionally, a fault message may also be defined.

3.2.4.2.1Document messages

Document message parts are composed from Schema element definitions (see R2204)

```
<wsdl:message …>
        <wsdl:part name=”..” element=”..”>
        <wsdl:part name=”..” element=”..”/>
</wsdl:message>
```

3.2.4.2.2RPC messages

RPC message parts are composed from Schema type declarations (see R2203)

```
<wsdl:message …>
        <wsdl:part name=” “ type=”..” />
        <wsdl:part name=” “ type=”..”/>
</wsdl:message>
```

Constraints on messages are listed in Section 5.13.

3.2.4.3 portTypes

The request/response transmission primitive must be used. Grammar is:

```
<wsdl:portType ..>
        <wsdl:operation …>
                <wsdl:input …/>
                <wsdl:output …/>
        </wsdl:operation>
</wsdl:portType>
```

Constraints on portTypes are listed in Section 5.14.

3.2.4.4 binding

The `wsdl:binding` section must use the SOAP binding extension with HTTP transport. The same operation defined in `wsdl:portType` must be used in the binding section.

```
<wsdl:binding …>
        <soap:binding style="rpc|document" transport=http://schemas.xmlsoap.org/soap/http>
                <wsdl:input …>
                </wsdl:input>
                <wsdl:output …>
                </wsdl:output>
        </soap:binding>
</wsdl:binding>
```

Constraints on bindings are listed in Section 5.15.

3.2.4.5 port

The `soap:address` element must be specified along with the URL for the endpoint:

```
<wsdl:port>
        <soap:address location="uri" />
</wsdl:port>
```

Constraints on port definitions are listed in Section 5.16.

3.2.5 UDDI

Advertisement of Web services patterned after this scenario adheres to the "Using WSDL in a UDDI Registry, Version 1.07" Best Practice document. A `uddi:tModel` representing the Web service type references the file containing the wsdl:binding for the synchronous message operation(s). The `uddi:bindingTemplate` captures the service endpoint and references the `uddi:tModel`(s) for the Web service type.

Advertising Web services in this way enables discovery using the inquiry patterns supported by the UDDI Inquiry API set (see http://www.uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf). These include the browse pattern, the drill-down pattern and the invocation pattern.

General UDDI Constraints are listed in Section 5.17.

3.2.6 Security

This section identifies the threats most relevant to this Usage Scenario as described in Appendix 1 where additional information on Basic Profile compliant countermeasures may also be found.

Additional constraints may apply if HTTPS is used to implement security. These are Basic Profile requirements: R5000, R5001, R5010. Appendix 1 has additional guidelines on Security.

As of this writing no specific threat has been identified as being singularly relevant to this Usage Scenario.

3.3 Basic Callback

3.3.1 Description

The Basic Callback scenario facilitates a form of asynchronous message exchange for Web services. This is accomplished through the composition of two synchronous request/response pairs. The messages are related via correlation information that is provided by the Consumer. The Consumer also provides the endpoint information for the callback service location to the Provider. The definition of the callback service is defined by the Provider in the published Web service description and implemented by the Consumer.

At runtime a Consumer sends the initial SOAP request in a request/response sequence to the Provider, which in turn sends back an immediate acknowledgement of receipt. At a later point in time the Provider will initiate the final request/response sequence to the Consumer containing the response data for the initial request sent by the Consumer. The following diagram shows the high-level interactions between a Consumer and a Provider in the Basic Callback Usage Scenario.

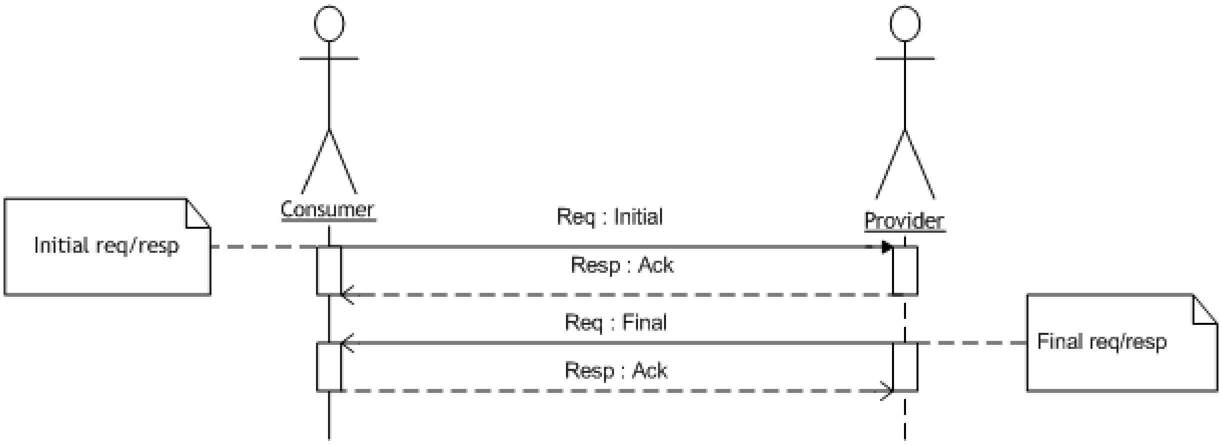

Figure 3-7 Basic Callback Sequence

<u>High-level flow</u>:

1. Consumer initiates the service by sending a SOAP message bound to an HTTP request to the Provider (the "initial request")
2. Provider acknowledges receipt via a SOAP message bound to an HTTP response to the Consumer (the "initial response")
3. Provider completes the exchange by sending a SOAP message bound to an HTTP request to the Consumer with the results of the initial request (the "final request" or "callback")
4. Consumer acknowledges receipt of the callback message with a SOAP message bound to an HTTP response (the "final response")

<u>Assumptions:</u>

- This scenario is a runtime sequence of events; it does not involve any design or deployment activities.
- The data model, the application semantics, the callback correlation mechanism, and the transport bindings are all agreed upon and implemented a priori to this scenario.
- All parts of this scenario are defined in conformance with the guidelines and recommendations of the Basic Profile.
- This scenario is "composable", that is, it may be used as a foundation for creating more complex scenarios.

3.3.2 Details

The Basic Callback scenario is built upon two correlated synchronous request/response interactions. Since a Consumer and a Provider may have many outstanding requests, there needs to be a mechanism for each party to unambiguously identify which callback goes with which initial request. This can be achieved using some business data in the SOAP payload, such as a purchase order number, that can be used to correlate the callback with the request, or through using some form of message id.

In order to invoke the callback service, the Consumer must communicate the callback endpoint to the Provider. This can be conveyed at runtime in the initial SOAP message sent by the Consumer to the Provider, during deployment, or using a discovery mechanism agreed to by both parties.

The Web service description for both the initial and final request/response pairs (i.e., portTypes) may be defined in a single WSDL document. Although it is not a requirement to do so, placing them in the same document will communicate the contract and the expectations on the client more effectively. In loosely coupled situations where two businesses may not want to maintain a single document, the initial and final request/response pairs should be described in separate WSDL documents. In either case, portTypes for both the Provider and Consumer Web services shall be defined. The description MAY contain ports for the Provider Web services, and DOES NOT contain defined ports for the Consumer Web services. Since the final service address is not known beforehand, a WSDL port cannot be defined for the final request/response portType. It is, instead communicated by the Consumer as described above.

3.3.3 Flow

The detailed flow for this scenario, using the activities defined in Section 2.2, is described below. Each bulleted item represents the activities performed within one layer of the stack required to complete the flow. The order of activities within a Consumer or Provider is not significant. Each activity has constraints imposed upon it from the Basic Profile.

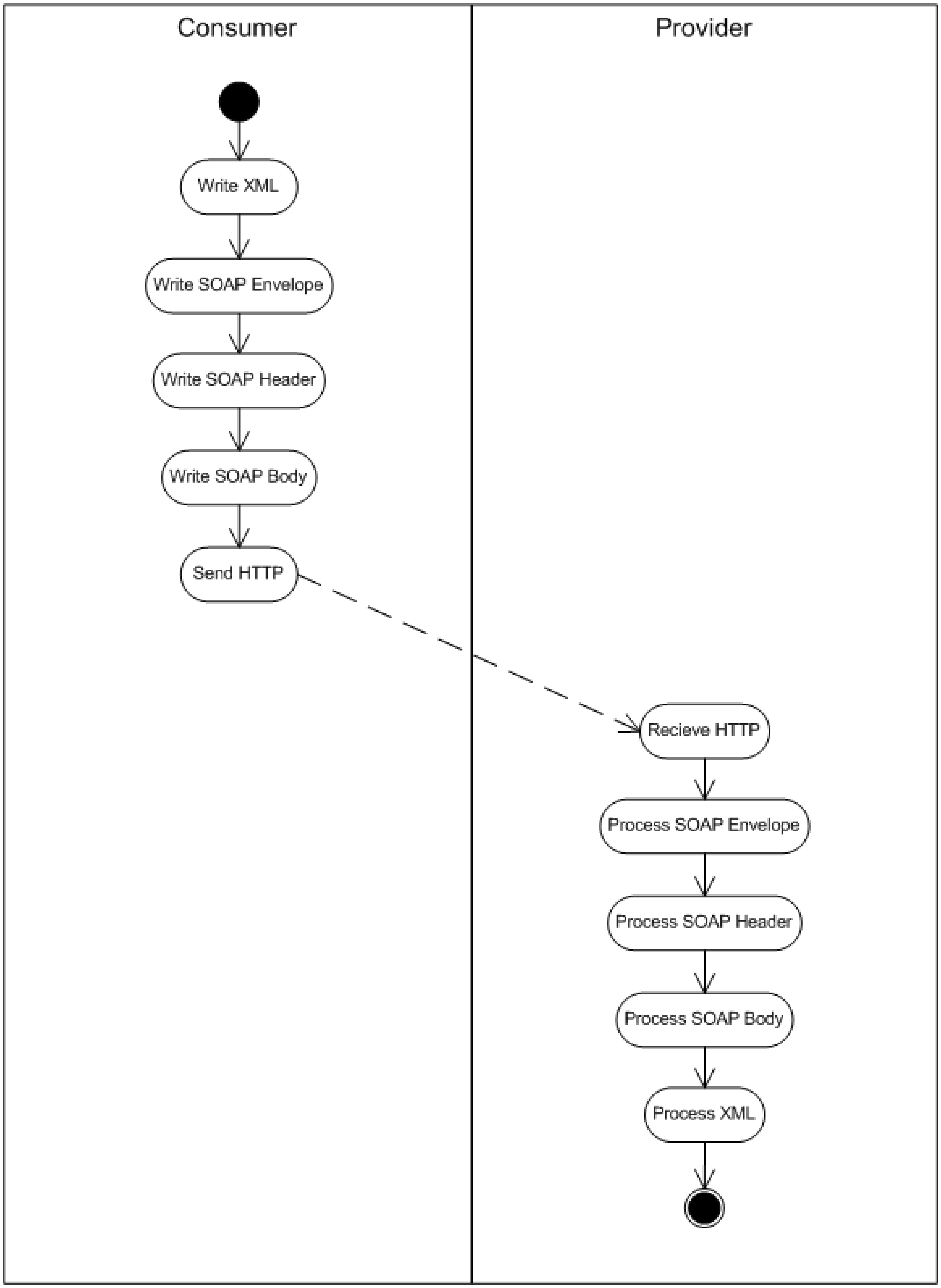

Figure 3-8 Basic Callback Consumer Request

The Consumer initiates a SOAP request:

- Data Layer
    - Write XML. The payload is created according to the data model.
- SOAP Message Layer
    - Write SOAP envelope
    - [Write SOAP header (if correlation information is conveyed in this manner)]
    - Write SOAP body
- Transport Layer
    - Send HTTP

The Provider receives the initial SOAP request:

- Transport Layer
    - Receive HTTP
- SOAP Message Layer
    - Process SOAP envelope
    - [Process SOAP header (if correlation information is conveyed in this manner)]
    - Process SOAP body
- Data Layer
    - Process XML.   The data payload is processed according to the data model and dispatched to the application

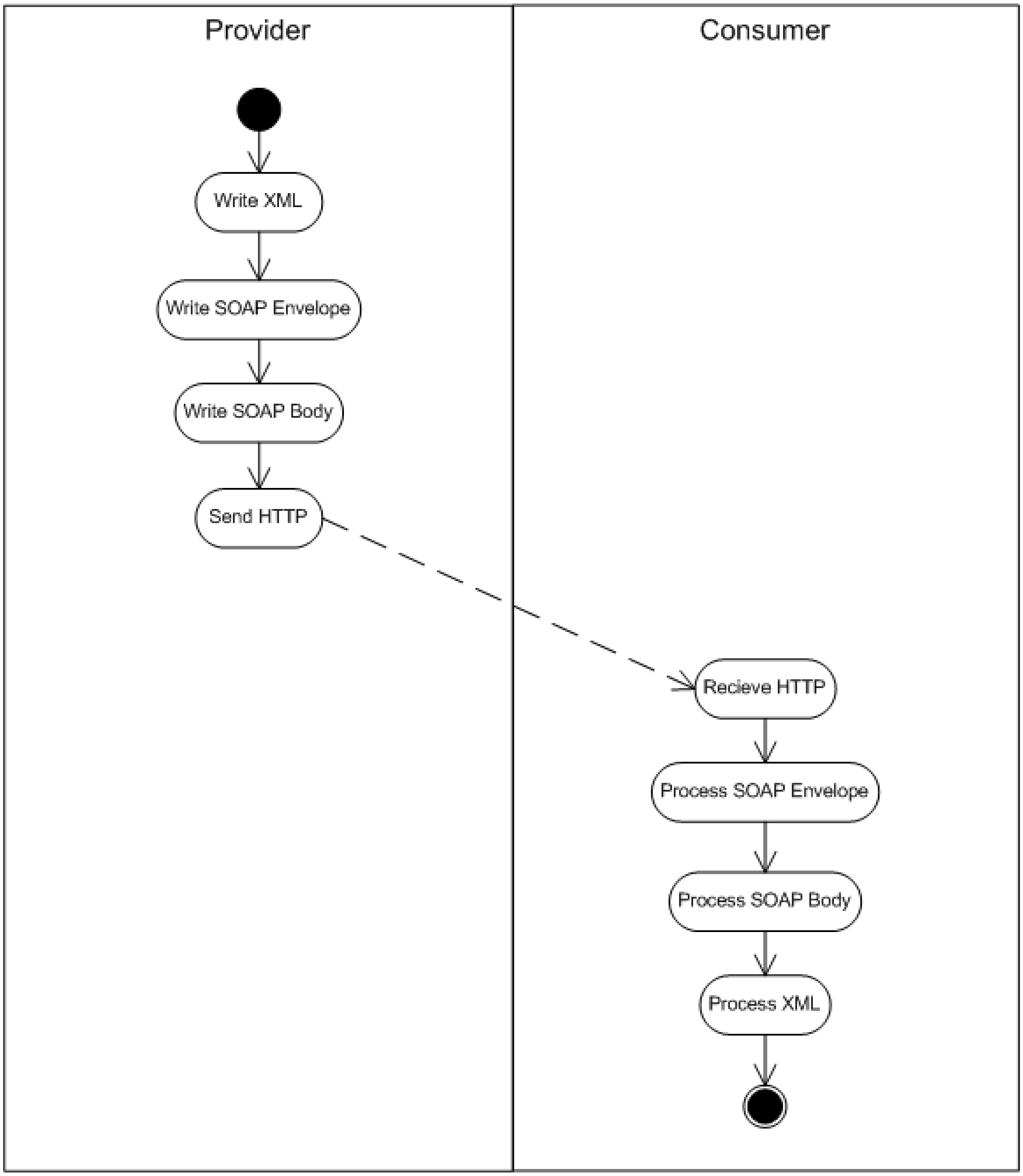

Figure 3-9 Basic Callback Provider Acknowledgement

The Provider generates the acknowledgement (response) message:

- Data Layer
  - Write XML. The payload is created according to the data model.
- SOAP Message Layer
  - Write SOAP envelope
  - Write SOAP body
- Transport Layer

    - Send HTTP

The Consumer receives the acknowledgement (response) message:

- Transport Layer
    - Receive HTTP
- SOAP Message Layer
    - Process SOAP envelope
    - Process SOAP body
- Data Layer
    - Process XML. The data payload is processed according to the data model and dispatched to the application

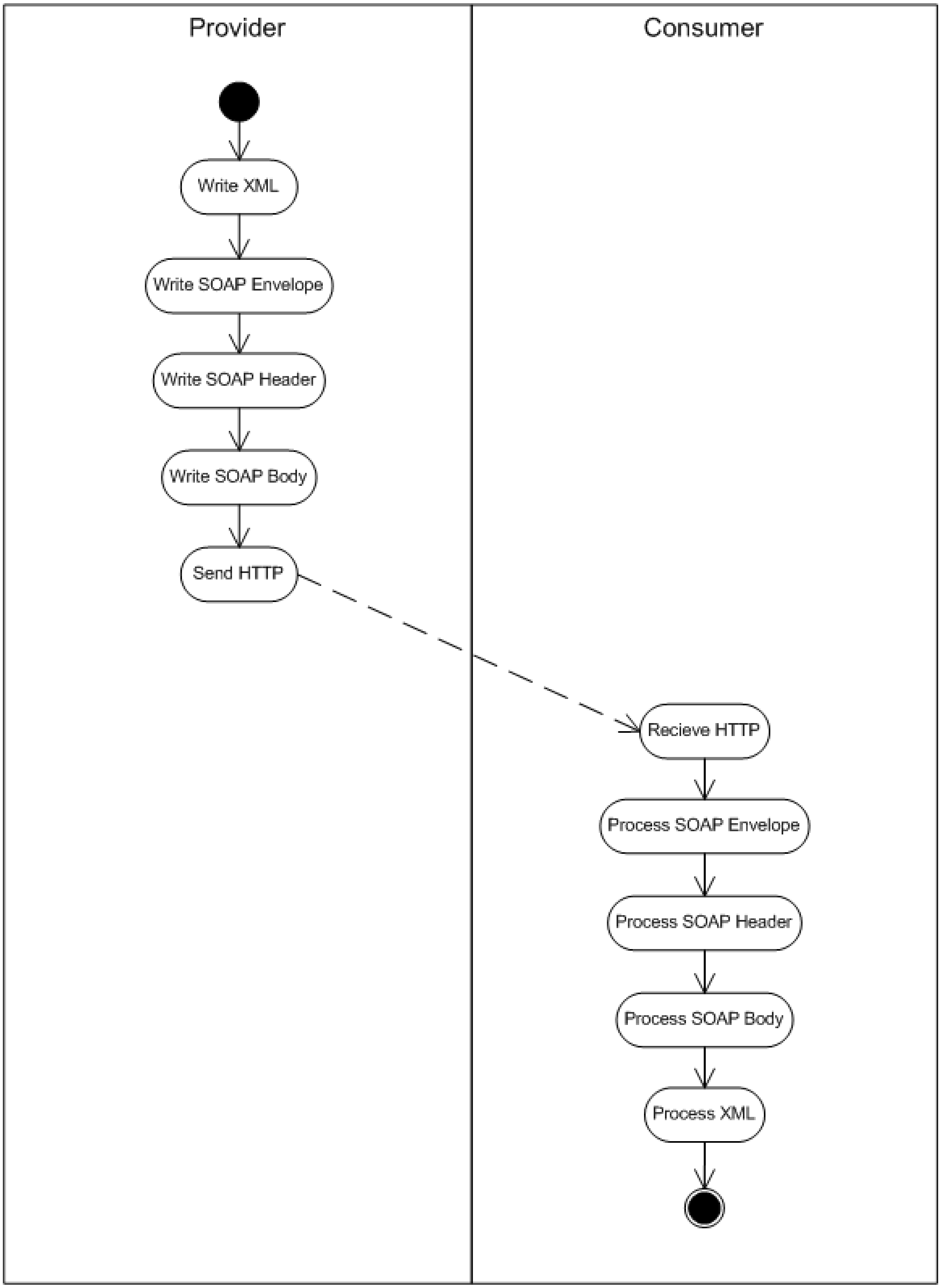

Figure 3-10 Basic Callback Provider Response

The Provider initiates a SOAP request:

- Data Layer
    - Write XML. The payload is created according to the data model.
- SOAP Message Layer
    - Write SOAP envelope
    - [Write SOAP header (if correlation information is conveyed in this manner)]
    - Write SOAP body
- Transport Layer
    - Send HTTP

The Consumer receives the SOAP request:

- Transport Layer
    - Receive HTTP
- SOAP Message Layer
    - Process SOAP envelope
    - [Process SOAP header (if correlation information is conveyed in this manner)]
    - Process SOAP body
- Data Layer
    - Process XML. The data payload is processed according to the data model and dispatched to the application

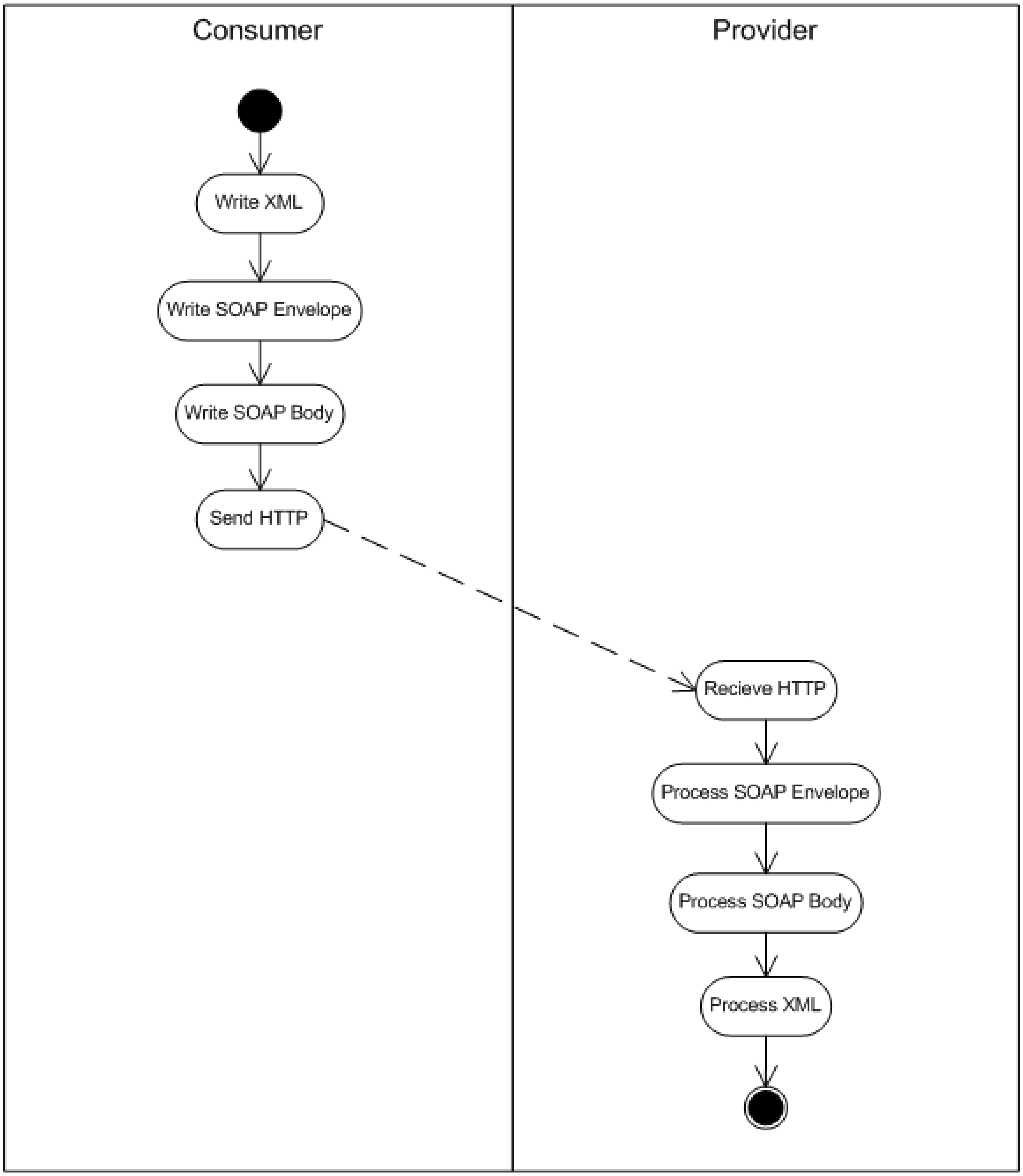

Figure 3-11 Basic Callback Consumer Acknowledgement

The Consumer then generates the acknowledgement (response) message:

- Data Layer
    - Write XML. The payload is created according to the data model.
- SOAP Message Layer
    - Write SOAP envelope
    - Write SOAP body
- Transport Layer

    - Send HTTP

The Provider receives the SOAP acknowledgement (response) message:

- Transport Layer
    - Receive HTTP
- SOAP Message Layer
    - Process SOAP envelope
    - Process SOAP body
- Data Layer
    - Process XML. The data payload is processed according to the data model and dispatched to the application

3.3.4 Flow Constraints

The following activities have the referenced constraints in this Usage Scenario.

- Write XML, as defined in Section 5.1
- Write SOAP envelope, as defined in Section 5.3
- [Write SOAP header (when correlation information is conveyed in a SOAP header)], as defined in Section 5.7
- Write SOAP body, as define in Section 5.5
- Send HTTP, as defined in Section 5.9
- Receive HTTP, as defined in Section 5.10
- Process SOAP envelope, as defined in Section 5.4
- [Process SOAP header (when correlation information is conveyed in a SOAP header)], as defined in Section 5.8
- Process SOAP body, as defined in Section 5.6
- Process XML, as defined in Section 5.2

3.3.4.1 Errors and SOAP Faults

Constraints for Fault generation and behavior as described in the Synchronous Request / Response Scenario 3.2.3.1 also apply to this scenario.

3.3.5 Description Constraints

For the Basic Callback scenario, the WSDL must have at least the following content within its definitions. Not all sections are required, but if present in the WSDL, each should follow the guidelines as presented below. The WSDL defined below is contained within a single document, and describes both the Initial and the Final request/response sequences of the Basic Callback. Constraints imposed upon the WSDL by the Basic Profile are also listed.

General constraints on the WSDL are described in Section 5.11. Other constraints imposed upon the WSDL by the Basic Profile are listed below.

3.3.5.1 types

Application Data

This section will be dependent upon the specifics of the data model and often contains correlation information in addition to application data types.

3.3.5.2 messages

Message format will be dependent upon the data model (doc/literal or rpc/literal). The following messages and parts are typically defined for this Usage Scenario:

- Initial request message
- Initial response message
- Final request message
- Final response message

3.3.5.2.1Document and RPC style

Below are some general issues concerning differences between Document and RPC style messages. For simplicity all required messages for this scenario have been defined as Document style, but this is not a requirement.

Document messages

Document message parts are composed from Schema element definitions (see R2204)

```
<wsdl:message …>
        <wsdl:part name=”InitialRequest” element=”..”>
</wsdl:message>
```

RPC messages

RPC message parts are composed from Schema type declarations (see R2203)

However, parts to be used in SOAP headers or faults MUST be defined as elements

```
<wsdl:message …>
        <wsdl:part name=”InitialRequest“ type=”..” />
</wsdl:message>
```

Constraints on messages are listed in Section 5.13.

3.3.5.3 portTypes

The request/response transmission primitive must be used for both the Initial and Final sequences.

3.3.5.3.1Provider

```
<wsdl:portType name="ProviderPortType">
        <wsdl:operation name="…">
                <wsdl:input…
                </wsdl:input>
                <wsdl:output…
                </wsdl:output>
        </wsdl:operation>
```

```
</wsdl:portType>
```

3.3.5.3.2Consumer

```
<wsdl:portType name="ConsumerPortType">
        <wsdl:operation name="…">
                <wsdl:input…
                </wsdl:input>
                <wsdl:output…
                </wsdl:input>
        </wsdl:operation>
</wsdl:portType>
```

Constraints on portTypes are listed in Section 5.14.

3.3.5.4 binding

The `wsdl:binding` section must use the SOAP binding extension with HTTP transport. The same operation defined in `wsdl:portType` must be used in the binding section. Two bindings will be defined, one for the Initial sequence, one for the Final.

3.3.5.4.1Provider

```
<wsdl:binding name="ProviderSoapBinding" type="tns:ProviderPortType">
<soap:binding style="document|rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="…">
        <soap:operation/>
        <wsdl:input…
        </wsdl:input>
        <wsdl:output…
        </wsdl:output>
</wsdl:operation>
</wsdl:binding>
```

3.3.5.4.2Consumer

```
<wsdl:binding name="ConsumerSoapBinding" type="tns:ConsumerPortType">
<soap:binding style="document|rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="submitFinalReq">
        <soap:operation/>
        <wsdl:input…
        </wsdl:input>
        <wsdl:output…
        </wsdl:output>
</wsdl:operation>
</wsdl:binding>
```

Constraints on bindings are listed in Section 5.15.

3.3.5.5 port

Only one port will be defined: that for the Initial Request/Response. The Final sequence has an unspecified port-binding. The soap:address element must be specified along with the URL for the endpoint:

```
<wsdl:port>
        <soap:address location=”uri” />
</wsdl:port>
```

Constraints on port definitions are listed in Section 5.16.

3.3.6 UDDI

There are two Web service implementations involved in this Usage Scenario, but only the Initial Web service is advertised in UDDI. The Final Web service is not discoverable because the callback endpoint must be known to and accessible by the initiator of the Initial request. Communicating the callback endpoint in the initial request accomplishes this.

Advertisement of Web services patterned after the Initial sequence in this scenario adheres to the “Using WSDL in a UDDI Registry, Version 1.07” Best Practice document. A uddi:tModel representing the Web service type references the containing document for the wsdl:binding of the Initial operation. The wsdl:binding corresponding to this portion of the Web service type is referenced using an xpointer based fragment identifier appended to the WSDL file URL. The uddi:bindingTemplate for the Initial sequence captures the service endpoint and references the uddi:tModel(s) for the Web service type.

Because the Final sequence in this scenario occurs between the two parties that participate in the Initial sequence, no advertisement or discovery of this sequence is desired or necessary.

Advertising Basic Callback Web services in this way enables discovery using the inquiry patterns supported by the UDDI Inquiry API set (see http://www.uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf). These include the browse pattern, the drill-down pattern and the invocation pattern.

General UDDI Constraints are listed in Section 5.17.

3.3.7 Security

This section identifies the threats most relevant to this Usage Scenario as described in Appendix 1 where additional information on Basic Profile compliant countermeasures may also be found.

Additional constraints may apply if HTTPS is used to implement security. These are Basic Profile requirements: R5000, R5001, R5010. Appendix 1 has additional guidelines on Security.

As of this writing no specific threat has been identified as being singularly relevant to this Usage Scenario, though Replay is being investigated as a potential candidate.

4 Appendix 1 – Security

This section details common Web service threats and suggests possible countermeasures that are compliant with the Basic Profile. The countermeasures detailed here are best applied through a risk assessment of your Web service application.

This information presented here is not intended to be an exhaustive or encyclopedic treatment of the security issues confronting Web services developers. Rather, it is designed to provide an intermediate assessment of security issues that briefly explores the intersection between traditional security issues and their manifestation in the Web services architecture.

4.1 Authentication

Authentication is a mechanism or a protocol that demonstrates proof of an asserted identity. Using an authentication mechanism, a Web service can draw conclusions about the sender of a request or response message, and then act on the message. Many types of authentication mechanisms and protocols have been developed, including password schemes, Secure Sockets Layer (SSL), Kerberos, and public key infrastructure. Each mechanism has advantages and limitations. With respect to interoperability, each mechanism introduces a variety of challenges. Web services can usually rely on the software platform to provide interoperable transport authentication. Additionally, Web services may wish to share authentication information across domains to provide single sign-on within a community of cooperating business entities.

Authentication requirements are usually asymmetrical between service requestors and service providers. Therefore, authentication for Web services can be further subdivided as below.

4.1.1 Request Authentication

Threat

The threats to a Web service that does not authenticate users include access to data or resources by unauthorized entities, and 'man-in-the-middle attacks'. In man-in-the-middle attacks, an unauthorized entity intercepts messages between requestor and responder, enabling eavesdropping and data manipulation. In very sophisticated Web services environments, a Web service provider may not be able to authenticate all parties involved in a transaction, and may therefore be required to delegate trust to other Web services.

Since Web services may rely on directory services to find providers of services (such as UDDI), authentication must be ensured in certain processes such as consulting UDDI registries or downloading WSDL files. If authentication is not required by a directory, a relatively easy attack would be to falsify a WSDL file, causing reliant Web services to bind to improper ports.

Countermeasure

A Web service SHOULD authenticate the sender of a request. Some specific situations in which Web services should authenticate requests include those in which underlying state is changed, in which there is a charge for using the service, or where the information returned by the service is privileged.

Authentication of the service requester is the appropriate countermeasure. Client authentication can be performed using agreed upon digital certificates in the client authentication piece of an SSL/TLS exchange. The digital certificates exchanged during the SSL handshake must chain to a certificate authority agreed upon by both client and server.

4.1.2 Response Authentication

Threat

An attack on a service requester is one that interposes a false or ‘spoofed” service that supplies responses resembling those provided by the expected service. For example, a “man-in-the-middle” might substitute a false response message for a genuine response, leading to request/response mismatches.

Countermeasure

Authentication of the service is the appropriate countermeasure. An SSL/TLS connection can provide server authentication and is typically sufficient protection from Web service provider spoofing for point-to-point transactions.

4.2 Authorization

Authorization is the process of determining the capabilities granted to an entity by a service provider or another trusted entity. While authentication determines which entities can access a Web service, authorization determines which features of that Web service can be accessed by the authenticated entity. In some cases even authenticated entities must be restricted to a subset of functions provided by a Web service.

4.2.1 Request Authorization

Threat

Unauthorized access to computational resources or protected data.

Countermeasure

Apply authorization mechanisms. Web services requests are fulfilled based on the authorization assigned to a particular requestor by the service provider. A Web service may need to communicate its authorization requirements through a policy.

A simple Web service may have one authorization level: i.e., I will execute process X for any user authenticated using a recognized token. However, more sophisticated mechanisms may be required for Web services designed to service a range of consumers.

4.3 Confidentiality

Threat

A compromise of privileged information through unauthorized access. In a messaging environment (as opposed to a session environment) it is important to evaluate the message protection characteristics of a Web service, because a Web service may not know the ultimate destination or the full route of the data being sent. Intermediaries may be traversed and if the data is unprotected, might read the confidential contents of a message, or they might be able to deduce confidential information by the mere fact that a particular message (or a message of a certain type, or messages in a certain frequency) was sent.

Countermeasure

Encryption is the primary defense against a breach of confidentiality. How encryption is applied can vary widely. The SSL/TLS protocol encrypts messages for the duration of the session. . However, at each end-point, the message will be fully decrypted. An exception to this situation is SSL Proxy tunneling. In which a client proxy opens a connection to a secure server, copying data in both directions without intervening in the secure transaction.

There are ways to address the problem of end-to-end confidentiality while remaining compliant, though out of scope, of the Basic Profile. As an example, XML Encryption can be used to selectively encrypt elements or the entire message. There are many configurations, but one is to have the SOAP implementation encrypt the message payload, while leaving other information "in the clear" in the SOAP header.

4.4 Data Integrity

Threat

Loss of data integrity is the unauthorized modification of a request or response. The threat to Web services is the malicious alteration or the accidental corruption of data.

Countermeasure

Messages sent using SSL/TLS have guaranteed data integrity for the duration of the exchange.

Another technique compliant, yet out of scope, with the Basic Profile is the use of digital signatures and message digests to provide proof of data integrity using XML Digital Signature. These can be applied to complete XML messages, or to portions of XML documents according to the XML Digital Signature specification.

4.5 Replay

Threat

A basic attack on a Web service is an attempt to re-use a once valid message. Certain elements of a Web services message, such as a security token, can also be reused as part of a different message to give the impression of a valid request or response.

Countermeasure

Replay attacks can be addressed by using message timestamps and caching, and through the use of universally unique identifiers on all messages.

4.6 Logging and Auditing

One of the best countermeasures for any of the above security issues is a robust auditing/logging mechanism. In combination with authentication mechanisms, auditing and logging mechanisms can provide chains of evidence that permit runtime infractions of trust policies to be remedied by the offline trust infrastructure of business agreements and contractual law.

4.7 Other Risks

Like any networked application, Web services are exposed to standard network security vulnerabilities such as:

- Unauthorized users gaining direct access to network resources
- Virus or Trojan horse programs being transmitted within otherwise valid XML messages
- Misconfiguration or improper coordination of internal resources by a Web services provider.
- Exploitation of known weaknesses
- Denial of Service attacks

5 Appendix 2 – Constraints

This section provides a mapping of constraints listed in the Basic Profile to each of the flow activities identified in Section 2 and within each scenario. In carrying out each activity, the listed constraints should be consulted in the Basic Profile to check for compliance with the details of the constraint.

5.1 Write XML

- XML Representation of SOAP Messages: R4001, R1008, R1009, R1010, R1012, R1013

5.2 Process XML

- XML Representation of SOAP Messages: R4001, R1008, R1009, R1010, R1012, R1013, R1015, R1017

5.3 Write SOAP Envelope

- Envelope structure: R1011, R2714

5.4 Process SOAP Envelope

- Envelope requirements: R1011, R1015, R1028, R2714

5.5 Write SOAP Body

- XML Representation of SOAP Messages: R1005, R1006, R1007, R1011, R1014, R2735, R2737
- The SOAP Processing Model: R1025, R1029, R1030
- RPC messages: R2729

5.6 Process SOAP Body

- XML Representation of SOAP Messages: R1005, R1006, R1007, R1014, R1017, R1028, R1029, R1030

5.7 Write SOAP Header

- XML Representation of SOAP Messages: R4001, R1005, R1008, R1009, R1010, R1012, R1013,
- The SOAP Processing Model: R1027
- Using SOAP in HTTP: R1109
- Header blocks: R2738, R2739, R2751, R2752, R2753

5.8 Process SOAP Header

- XML Representation of SOAP Messages: R1012, R1005, R1008, R1009, R1010, R1012, R1013, R1015, R1017,
- The SOAP Processing Model: R1025, R1026, R1027, R1028, R1029, R1030,

5.9 Send HTTP

The following constraints apply to both HTTP and HTTPS.

- General: R1108, R1140, R1141, R1132
- Status code: R1106, R1107, R1111, R1112, R1113, R1114, R1115, R1116, R1124, R1125, R1126, R1130
- SOAPAction Header: R1109, R2713, R2744, R2745
- Cookies: R1120, R1121, R1122, R1123

5.10 Receive HTTP

The following constraints apply to both HTTP and HTTPS.

- General: R1110, R1140, R2746
- Status code: R1107, R1111, R1112, R1113, R1114, R1115, R1116, R1124, R1125, R1126, R1130, R1131
- SOAPAction Header: R1119
- Cookies: R1120, R1121, R1122, R1123

5.11 General WSDL Constraints

- Description of an Instance: R0001
- Importing documents into WSDL: R2001, R2002, R2003, R2004, R2005, R2007, R2008, R2009, R2010, R2011
- Constraints on the overall structure of WSDL: R2020, R2021, R2022, R2023, R2024, R2025, R2026, R2027, R2028, R4002, R4003, R4004
- WSDL Extensions: R2747, R2748

5.12 Constraints on WSDL types

- Constraint on use of QNames: R2101, R2102
- Constraint on declaration of array types: R2110, R2111, R2112, R2113
- Usage of XML Schema: R2105, R2114, R2800, R2801

5.13 Constraints on WSDL messages

- Constraints relating to bindings and parts: R2201, R2202, R2203, R2204, R2206, R2207, R2208, R2210
- Constraints on portType: R2209

5.14 Constraints on WSDL portTypes

- Wire representation of the message: R2301, R2302, R2305, R2306, R2710, R2712
- Constraints on operations: R2303, R2304

5.15 Constraints on WSDL Bindings

- Structure: R2029
- Allowed bindings: R2401, R2700
- Transport constraints: R2701, R2702
- Constraints on soap:style: R2705
- Constraints on soap:use: R2706, R2707
- Relationship to portTypes: R2709, R2718
- Using SOAPAction: R2713
- Using soap:namespace attribute: R2716, R2717, R2726

- Faults and header constraints: R2719, R2720, R2721, R2722, R2723, R2740, R2741, R2749

5.16 Constraints on WSDL Port

- Allowed bindings: R2711

5.17 General UDDI constraints

- Description of an Instance: R0001
- Constraints on binding templates: R3100
- Constraints on tModels: R3002, R3003, R3005, R3010, R3011

6 References

[1] WS-I Basic Profile version 1.0 from www.ws-i.org.