Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions

W3C Working Draft 3 August 2004

This version:
http://www.w3.org/TR/2004/WD-wsdl20-extensions-20040803
Latest version:
http://www.w3.org/TR/wsdl20-extensions
Previous versions:
http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326
Editors:
Martin Gudgin, Microsoft
Amy Lewis, TIBCO
Jeffrey Schlimmer, Microsoft

This document is also available in these non-normative formats: postscript, PDF, XML, and plain text.

Copyright Â© 2004 W3CÂ® (MIT, ERCIM, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This document describes extensions for the Web Services Description Language (WSDL) Version 2.0 . These extensions include Message Exchange Patterns (MEPs), features, SOAP modules, and bindings of features. The Working Group has discussed and approved these extensions, and recommends their use with the Web Services Description Language (WSDL).

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is a W3C Last Call Working Draft. If the feedback is positive, the Working Group plans to submit this specification for consideration as a W3C Candidate Recommendation. Comments on this document are invited and are to be sent to the public public-ws-desc-comments@w3.org mailing list (public archive). Comments can be sent until **4 October 2004**.

Three formal objections from Working Group participants have been received against portions of the WSDL 2.0 specification. Feedback is specifically encouraged on these topics:

- Compositors (see objection)
- Feature and properties (see objection and follow-on message)
- Requiring unique GEDs or required feature to distinguish operations (see objection)

A diff-marked version against the previous version of this document is available. For a detailed list of changes since the last publication of this document, please refer to appendix **B. Change Log** [p.13] . Issues about this document are documented in the last call issues list maintained by the Working Group.

This document has been produced as part of the W3C Web Services Activity. The authors of this document are the Web Services Description Working Group members.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document has been produced under the 24 January 2002 Current Patent Practice as amended by the W3C Patent Policy Transition Procedure. Patent disclosures relevant to this specification may be found on the Working Group's patent disclosure page. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with section 6 of the W3C Patent Policy.

Short Table of Contents

1. Introduction [p.3]
2. Predefined Message Exchange Patterns [p.4]
3. Predefined Features [p.8]
4. References [p.12]
A. Acknowledgements [p.12] (Non-Normative)
B. Change Log [p.13] (Non-Normative)

Table of Contents

1. Introduction [p.3]
  1.1 Notational Conventions [p.3]
2. Predefined Message Exchange Patterns [p.4]
  2.1 Fault Propagation Rules [p.4]
    2.1.1 Fault Replaces Message [p.4]
    2.1.2 Message Triggers Fault [p.5]
    2.1.3 No Faults [p.5]
  2.2 Message Exchange Patterns [p.5]

2.2.1 In-Only [p.5]
2.2.2 Robust In-Only [p.5]
2.2.3 In-Out [p.6]
2.2.4 In-Optional-Out [p.6]
2.2.5 Out-Only [p.6]
2.2.6 Robust Out-Only [p.7]
2.2.7 Out-In [p.7]
2.2.8 Out-Optional-In [p.8]
3. Predefined Features [p.8]
3.1 Application Data Feature [p.8]
3.1.1 Name [p.9]
3.1.2 Operation [p.9]
3.1.3 AD/data Property [p.9]
3.1.4 Description [p.9]
3.1.5 The ad:mustUnderstand Attribute [p.10]
3.2 Application Data Module [p.10]
3.2.1 Features Implemented [p.10]
3.2.2 Operation [p.10]
3.3 Application data serialization as HTTP headers [p.11]
3.3.1 Name [p.11]
3.3.2 Features Implemented [p.11]
3.3.3 Operation [p.11]
4. References [p.12]
4.1 Normative References [p.12]
4.2 Informative References [p.12]

Appendices

A. Acknowledgements [p.12] (Non-Normative)
B. Change Log [p.13] (Non-Normative)
B.1 Changes [p.13]

---

1. Introduction

Web Services Description Language provides a number of opportunities to extend the syntax and component model, as mandated by the needs of an application. This document defines and describes a number of these extensions, particularly message exchange patterns and features.

1.1 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 [*IETF RFC 2119 [p.12]*].

2. Predefined Message Exchange Patterns

Web Services Description Language (WSDL) message exchange patterns (hereafter simply 'patterns') define the sequence and cardinality of abstract messages listed in an operation. Message exchange patterns also define which other nodes send messages to, and receive messages from, the service implementing the operation. WSDL message exchange patterns describe the interaction at the abstract (interface) level, which may be distinct from the pattern used by the underlying protocol binding (e.g. SOAP Message Exchange Patterns).

By design, WSDL message exchange patterns abstract out specific message types. Patterns identify placeholders for messages, and placeholders are associated with specific message types by the operation using the pattern.

Unless explicitly stated otherwise, WSDL message exchange patterns also abstract out binding-specific information like timing between messages, whether the pattern is synchronous or asynchronous, and whether the message are sent over a single or multiple channels.

Like interfaces and operations, WSDL message exchange patterns do not exhaustively describe the set of messages exchanged between a service and other nodes; by some prior agreement, another node and/or the service may send other messages (to each other or to other nodes) that are not described by the pattern. For instance, even though a pattern may define a single message sent from a service to one other node, the Web Service may multicast that message to other nodes.

To maximize reuse, WSDL message exchange patterns identify a minimal contract between other parties and Web Services, and contain only information that is relevant to both the Web Service and another party.

This specification defines several message exchange patterns for use with *WSDL Version 2.0 Part 1: Core Language* [*WSDL 2.0 Core Language [p.12]*].

2.1 Fault Propagation Rules

WSDL patterns specify their fault propagation model using standard rulesets to indicate where faults may occur. The most common patterns for fault propagation are defined here, and referenced by patterns later in the document.

Generation of a fault, regardless of ruleset, terminates the exchange.

2.1.1 Fault Replaces Message

Any message after the first in the pattern MAY be replaced with a fault message, which MUST have identical cardinality and direction. The fault message MUST be delivered to the same target node as the message it replaces. If there is no path to this node, the fault MUST be discarded.

2.1.2 Message Triggers Fault

Any message, including the first, MAY trigger a fault message in response. Each recipient MAY propagate a fault message, and MUST propagate no more than one fault for each triggering message. Each fault message has direction the reverse of its triggering message. The fault message MUST be delivered to the originator of the message which triggered it. If there is no path to this node, the fault MUST be discarded.

2.1.3 No Faults

No faults may be propagated.

2.2 Message Exchange Patterns

WSDL patterns are described in terms of the WSDL component model, specifically the Message Label and Fault Reference components.

2.2.1 In-Only

This pattern consists of exactly one message as follows:

1. A message:
    - indicated by a Message Label component whose {message label} is 'In' and {direction} is 'in'
    - received from some node N

This pattern uses the rule **2.1.3 No Faults** [p.5] .

An operation using this message exchange pattern has a {pattern} property with the value 'http://www.w3.org/2004/08/wsdl/in-only'.

2.2.2 Robust In-Only

This pattern consists of exactly one message as follows:

1. A message:
    - indicated by a Message Label component whose {message label} is 'In' and {direction} is 'in'
    - received from some node N

This pattern uses the rule **2.1.2 Message Triggers Fault** [p.5] .

An operation using this message exchange pattern has a {pattern} property with the value 'http://www.w3.org/2004/08/wsdl/robust-in-only'.

2.2.3 In-Out

This pattern consists of exactly two messages, in order, as follows:

1. A message:
   - indicated by a Message Label component whose {message label} is 'In' and {direction} is 'in'
   - received from some node N
2. A message:
   - indicated by a Message Label component whose {message label} is 'Out' and {direction} is 'out'
   - sent to node N

This pattern uses the rule **2.1.1 Fault Replaces Message** [p.4] .

An operation using this message exchange pattern has a {pattern} property with the value 'http://www.w3.org/2004/08/wsdl/in-out'.

2.2.4 In-Optional-Out

This pattern consists of one or two messages, in order, as follows:

1. A message:
   - indicated by a Message Label component whose {message label} is 'In' and {direction} is 'in'
   - received from some node N
2. An optional message:
   - indicated by a Message Label component whose {message label} is 'Out' and {direction} is 'out'
   - sent to node N

This pattern uses the rule **2.1.2 Message Triggers Fault** [p.5] .

An operation using this message exchange pattern has a {pattern} property with the value 'http://www.w3.org/2004/08/wsdl/in-opt-out'.

2.2.5 Out-Only

This pattern consists of exactly one message as follows:

1. A message:
    - indicated by a Message Label component whose {message label} is 'Out' and {direction} is 'out'
    - sent to some node N

This pattern uses the rule **2.1.3 No Faults** [p.5] .

An operation using this message exchange pattern has a {pattern} property with the value 'http://www.w3.org/2004/08/wsdl/out-only'.

2.2.6 Robust Out-Only

This pattern consists of exactly one message as follows:

1. message:
    - indicated by a Message Label component whose {message label} is 'Out' and {direction} is 'out'
    - sent to some node N

This pattern uses the rule **2.1.2 Message Triggers Fault** [p.5] .

An operation using this message exchange pattern has a {pattern} property with the value 'http://www.w3.org/2004/08/wsdl/robust-out-only'.

2.2.7 Out-In

This pattern consists of exactly two messages, in order, as follows:

1. A message:
    - indicated by a Message Label component whose {message label} is 'Out' and {direction} is 'out'
    - sent to some node N
2. A message:
    - indicated by a Message Label component whose {message label} is 'In' and {direction} is 'in'
    - sent from node N

This pattern uses the rule **2.1.1 Fault Replaces Message** [p.4] .

An operation using this message exchange pattern has a {pattern} property with the value ’http://www.w3.org/2004/08/wsdl/out-in’.

2.2.8 Out-Optional-In

This pattern consists of one or two messages, in order, as follows:

1. A message:
    - indicated by a Message Label component whose {message label} is ’Out’ and {direction} is ’out’
    - sent to some node N
2. An optional message:
    - indicated by a MessageLabel component whose {message label} is ’In’ and {direction} is ’in’
    - sent from node N

This pattern uses the rule **2.1.2 Message Triggers Fault** [p.5] .

An operation using this message exchange pattern has a {pattern} property with the value ’http://www.w3.org/2004/08/wsdl/out-opt-in’.

3. Predefined Features

Web Services Description Language (WSDL) features (hereafter ’features’) define pieces of extended functionality which typically affect message exchanges. Examples may include "reliability", "security", or "correlation", among others. Features may be self-contained, or may be abstract, and thus expressed via bindings or SOAP modules. These components may expose settable properties, named variables which affect the behavior of one or more features, bindings or SOAP modules.

Features may change the behavior described for other components. In particular, note that a feature (or any other extension) may change the semantics of a message exchange pattern in some fashion, such as nominating an address for the delivery of faults, etc.

The Web Services Description Working Group provides the following predefined features and SOAP modules for two reasons - first, we encourage implementors to support these features as we believe they offer important functionality. Second, these specifications act as a model of how to write feature/module specs. Some further (SOAP-specific) examples can be found in the SOAP 1.2 spec, part 2.

3.1 Application Data Feature

3.1.1 Name

This feature is identified with the URI 'http://www.w3.org/2004/08/wsdl/feature/AD'

3.1.2 Operation

This feature exists in order to enable the description of application-defined additional data declarations outside of the normal data channel (e.g. the SOAP body). The senders takes the value of the property 'http://www.w3.org/2004/08/wsdl/feature/AD/data', which is defined below, and passes it to the receiver in a manner to be defined by the particular bindings/modules implementing this specification.

3.1.3 AD/data Property

This property is identified with the URI 'http://www.w3.org/2004/08/wsdl/feature/AD/data'.

3.1.4 Description

The data property consists of a sequence of elements, each of which represents an individual piece of application data. Implementations of this feature must ensure that the runtime value of this property is correctly transferred from the sender to the receiver.

Here is an example of using the data property in a WSDL:

```
<types>
 <schema targetNamespace="http://example.com/ws/wsdl20/my-ws"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:ad="http://www.w3.org/2004/08/wsdl/feature/AD">
  <!-- Define the data type we'll use later -->
  <complexType name="myDataType">
   <sequence>

    <!-- These elements are our data -->
    <element name="isGoldClubMember" type="xs:boolean"
             ad:mustUnderstand="true" />

    <element name="promotionalCode"
             type="xs:string"
             minOccurs="0"/>

   </sequence>
  </complexType>
 </schema>
</types>
<interface name="customerService">
 <operation name="reserveCar">
  <input element="myNS:reserveCarRequest">
   <property uri="http://www.w3.org/2004/08/wsdl/feature/AD/data">
    <constraint xmlns:foo="http://example.com/">
     foo:myDataType
    </constraint>
```

```
   </property>
  </input>
 </operation>
</interface>
```

This example defines two pieces of application data, and associates them with the input message of the "reserveCar" operation. Notice that the "promotionalCode" element is optional (minOccurs="0").

3.1.5 The `ad:mustUnderstand` Attribute

You may choose to decorate your application data element declarations with an attribute with the namespace 'http://www.w3.org/2004/08/wsdl/feature/AD' and the local name "mustUnderstand". This indicates at the abstract level that the particular element thus decorated is mandatory, and implementations of this feature which support expression of mandatory data (i.e. the Application Data SOAP Module, see **3.2 Application Data Module** [p.10]) should mark them as mandatory in an appropriate way.

3.2 Application Data Module

This module is identified with the URI 'http://www.w3.org/2004/08/wsdl/module/AD'

3.2.1 Features Implemented

This module implements the feature 'http://www.w3.org/2004/08/wsdl/feature/AD' (see **3.1 Application Data Feature** [p.8]).

3.2.2 Operation

This module specifies how to transmit "out of band" application data, as defined in the Application Data feature (see **3.1 Application Data Feature** [p.8]), as SOAP headers.

As a SOAP sender, if the property 'http://www.w3.org/2004/08/wsdl/feature/AD/data' has a value then each of the top-level child element information items in the value MUST be turned into a SOAP header. The elements are serialized according to their schemas, and if the "ad:mustUnderstand" attribute exists with the value "true" on any given element declaration, that particular SOAP header should be marked as "mustUnderstand='true'" or "mustUnderstand='1'" as per the SOAP specification. SOAP senders SHOULD also add an additional header, with namespace 'http://www.w3.org/2004/08/wsdl/module/AD' and local name "dataHeaders" - this header contains a list of element QNames, one for each application data header created in the first step.

It is the responsibility of the receiving node to determine which, if any, SOAP headers will populate the 'http://www.w3.org/2004/08/wsdl/feature/AD/data' property. Typically this will be accomplished via using some metadata, such as an understanding of a constraint specified in WSDL, or out-of-band agreements. If the "dataHeaders" SOAP header (described above) is present, the QNames inside that header indicate which other headers are application data. The contents of each SOAP header identified as application data will be placed in a child element of the data property.

3.3 Application data serialization as HTTP headers

3.3.1 Name

This feature-binding is identified with the URI ’http://www.w3.org/2004/08/wsdl/feature/AD-HTTP’

3.3.2 Features Implemented

This module implements the feature ’http://www.w3.org/2004/08/wsdl/feature/AD’ (see **3.1 Application Data Feature** [p.8]).

3.3.3 Operation

This section specifies how to transmit "out of band" application data, defined in the Application Data Feature (see **3.1 Application Data Feature** [p.8]) as HTTP headers.

If the property ’http://www.w3.org/2004/08/wsdl/feature/AD/data’ has a value for a message to be serialized as an HTTP message, then each of the top-level child *element information item*s indicates an *element information item* that MUST be turned into an HTTP header if possible.

Only *element information item*s of type "xs:string" or "xs:anyURI" may be serialized. All complex data types are ignored. Attributes on data elements are ignored.

Each such *element information item* is serialized as follows:

- The HTTP header name used is the *element information item* local name. The *element information item* local name MUST follow the field-name production rules as specified in section 4.2 of [*IETF RFC 2616 [p.12]*]; if not, the *element information item* MUST be ignored. If an HTTP header corresponding to the *element information item* local name is set by a mechanism other than the Application Data Feature (see **3.1 Application Data Feature** [p.8]), such as the HTTP stack or another feature, then an error MUST be raised.

- The HTTP header content is serialized from the *element information item* value in UTF-8. If this serialization is NOT possible, then the *element information item* MUST be ignored.

It is the responsibility of the receiving node to determine which, if any, HTTP headers will populate the ’http://www.w3.org/2004/08/wsdl/feature/AD/data’ property. Typically this will be accomplished via some metadata, such as a {property constraint} specified in [*WSDL 2.0 Core Language [p.12]*], or out-of-band agreements. The content of each such HTTP header will be placed in a child element of the data property. Each child *element information item* is generated by using the HTTP header name as the *element information item* local name and the HTTP header value as the *element information item* value.

Note:

The local name of the *element information item* which is the parent node of the *element information item*s received, as well as the namespace of those *element information item*s, are implementation-specific.

4. References

4.1 Normative References

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

[IETF RFC 2616]
Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Authors. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

[WSDL 2.0 Core Language]
Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, R.Chinnici, M.Gudgin, J-J. Moreau, S.Weerawarana Editors. World Wide Web Consortium, 3 August 2004. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" Specification is available is available at http://www.w3.org/TR/2004/WD-wsdl20-20040803. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" is available at http://www.w3.org/TR/wsdl20.

4.2 Informative References

[WSD Requirements]
Web Services Description Requirements, J. Schlimmer, Editor. World Wide Web Consortium, 28 October 2002. This version of the Web Services Description Requirements document is http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028. The latest version of Web Services Description Requirements is available at http://www.w3.org/TR/ws-desc-reqs.

A. Acknowledgements (Non-Normative)

This document is the work of the W3C Web Service Description Working Group.

Members of the Working Group are (at the time of writing, and by alphabetical order): David Booth (W3C), Allen Brookes (Rogue Wave Softwave), Helen Chen (Agfa-Gevaert N. V.), Roberto Chinnici (Sun Microsystems), Ugo Corda (SeeBeyond), Glen Daniels (Sonic Software), Paul Downey (British Telecommunications), Youenn Fablet (Canon), Martin Gudgin (Microsoft Corporation), Hugo Haas (W3C), Hao He (The Thomson Corporation), Tom Jordahl (Macromedia), Jacek Kopecky (Digital Enterprise Research Institute (DERI)), Amelia Lewis (TIBCO Software, Inc.), Kevin Canyang Liu (SAP), Jonathan Marsh (Microsoft Corporation), Peter Madziak (Agfa-Gevaert N. V.), Josephine Micallef (SAIC - Telcordia Technologies), Jeff Mischkinsky (Oracle Corporation), Dale Moberg (Cyclone Commerce), Jean-Jacques Moreau (Canon), Mark Nottingham (BEA Systems, Inc.), David Orchard (BEA Systems, Inc.), Bijan Parsia (University of Maryland), Arthur Ryman (IBM), Adi Sakala (IONA Technologies), Jeffrey Schlimmer (Microsoft Corporation), Igor Sedukhin (Computer Associates), Jerry Thrasher (Lexmark), William Vambenepe (Hewlett-Packard Company), Asir Vedamuthu (webMethods, Inc.), Sanjiva Weerawarana (IBM), Ãmit YalÃ§Ä±nalp (Oracle Corporation), Prasad Yendluri (webMethods, Inc.).

Previous members were: Lily Liu (webMethods, Inc.), Don Wright (Lexmark), Joyce Yang (Oracle Corporation), Daniel Schutzer (Citigroup), Dave Solo (Citigroup), Stefano Pogliani (Sun Microsystems), William Stumbo (Xerox), Stephen White (SeeBeyond), Barbara Zengler (DaimlerChrysler Research and Technology), Tim Finin (University of Maryland), Laurent De Teneuille (L'Echangeur), Johan Pauhlsson (L'Echangeur), Mark Jones (AT&T), Steve Lind (AT&T), Sandra Swearingen (U.S. Department of Defense, U.S. Air Force), Philippe Le HÃ©garet (W3C), Jim Hendler (University of Maryland), Dietmar Gaertner (Software AG), Michael Champion (Software AG), Don Mullen (TIBCO Software, Inc.), Steve Graham (Global Grid Forum), Steve Tuecke (Global Grid Forum), Michael Mahan (Nokia), Bryan Thompson (Hicks & Associates), Ingo Melzer (DaimlerChrysler Research and Technology), Sandeep Kumar (Cisco Systems), Alan Davies (SeeBeyond), Jacek Kopecky (Systinet), Mike Ballantyne (Electronic Data Systems), Mike Davoren (W. W. Grainger), Dan Kulp (IONA Technologies), Mike McHugh (W. W. Grainger), Michael Mealling (Verisign), Waqar Sadiq (Electronic Data Systems), Yaron Goland (BEA Systems, Inc.).

The people who have contributed to discussions on www-ws-desc@w3.org are also gratefully acknowledged.

B. Change Log (Non-Normative)

B.1 Changes

Date	Author	Description
20040713	aal	implement editorial changes requested after review by GlenD, in application data feature and module.
20040713	aal	address issues 233 & 112 all at once, by increasing level of all divs, adding new intro div, adding new div to contain features, renaming spec. Lotsa changes, what fun.
20040713	aal	s/Label/Message Label/g and s/{label}/{message label}/g. issue 230.
20040713	aal	replace "fault generation" with "fault propagation" (in almost all cases; one case of "generate" remains to indicate that it ends an exchange). issue 234.
20040713	aal	add language to introduction describing relationship between these MEPs and the MEPs defined by SOAP 1.2 (issue 232). This replaces the language found two items down (issue 191).
20040713	aal	add (hereafter, simply 'patterns') to intro (issue 231).
20040610	aal	add language to introduction describing relationship between these MEPs and the MEPs defined by SOAP 1.2 (issue 191).
20040225	aal	add in-optional-out per minutes of 20 feb 2004 telecon
20040212	aal	change {messageReference} to {label} and "Message Reference component" to "Label component" per 20040212 teleconference

20040205	aal	change all ’A’ and ’B’ message labels into ’Out’ or ’In’, depending upon direction.
20040205	aal	s/message pattern/message exchange pattern/gi
20031204	jcs	Removed change marks; note that some were on div2 tag and did not show when transformed into HTML.
20031204	jcs	Per 4 Dec 2003 telecon, decided to rename ’Asynchronous Out-In’ pattern to ’Output-Optional-Input’.
20031105	aal	Fix titles of added patterns. Move them to be in conjunction with similar patterns.
20031022	aal	Per action item from October 16 teleconference, added the three patterns using message-triggers-fault as published on the mailing list (robust-in-only, robust-out-only, asynch-out-in).
20031022	aal	Added internal linkage (using specref) from patterns to the fault rulesets which they use.
20031022	aal	Per 9 and 16 Oct 2003 teleconferences, marked in-multi-out and out-multi-in patterns deleted.
20031022	aal	Per 16 Oct 2003 teleconference, added a paragraph/sentence stating that generation of a fault terminates an exchange.
20031007	JCS	Per 2 Oct 2003 teleconference, changed "broadcast" to "multicast" in the introduction.
20030922	JCS	Per 22 Sep 2003 meeting in Palo Alto, CA, removed "Pattern Review" editorial note; added specific editorial notes for In-Multi-Out and Out-Multi-In.
20030911	RRC	Changed the "name" property of the message reference component to "messageReference".
20030904	JCS	Incorporated clarifications suggested by W3C\David Booth.
20030801	JCS	Per 30 July meeting, added recommendations from patterns task force.
20030612	AAL	Added fault generation rulesets and references to them from patterns.
20030313	MJG	Changed to Part 2 (from Part 3)
20030306	JCS	Proposed name for MEP7.
20030305	JCS	Per 4 Mar 03 meeting, renamed ’message exchange pattern’ to ’message pattern’ or ’pattern’, added pattern for request-response, added ednote about review of patterns.
20030217	MJG	Fixed some issues with entities and validity errors WRT ulists
20030212	JCS	Initial draft