

PPS (Production Planning and Scheduling) Part 2: Transaction Messages, Version 1.0

Public Review Draft 02

22 December 2007

Specification URIs:

<http://docs.oasis-open.org/pps/v1.0/pr02/pps-transaction-messages-1.0-pr02.doc>
<http://docs.oasis-open.org/pps/v1.0/pr02/pps-transaction-messages-1.0-pr02.html>
<http://docs.oasis-open.org/pps/v1.0/pr02/pps-transaction-messages-1.0-pr02.pdf>

Previous Version:

<http://docs.oasis-open.org/pps/v1.0/pr01/pps-transaction-messages-1.0-pr01.doc>
<http://docs.oasis-open.org/pps/v1.0/pr01/pps-transaction-messages-1.0-pr01.html>
<http://docs.oasis-open.org/pps/v1.0/pr01/pps-transaction-messages-1.0-pr01.pdf>

Latest Version:

<http://docs.oasis-open.org/pps/v1.0/pps-transaction-messages-1.0.doc>
<http://docs.oasis-open.org/pps/v1.0/pps-transaction-messages-1.0.html>
<http://docs.oasis-open.org/pps/v1.0/pps-transaction-messages-1.0.pdf>

Technical Committee:

OASIS Production Planning and Scheduling TC

Chair(s):

Yasuyuki Nishioka, PSLX Forum / Hosei University

Editor(s):

Yasuyuki Nishioka, PSLX Forum / Hosei University
Koichi Wada, PSLX Forum

Related work:

This specification is related to:

- Universal Business Language 2.0

Declared XML Namespace(s):

<http://docs.oasis-open.org/pps/ns>

Abstract:

OASIS PPS (Production Planning and Scheduling) Standard deals with problems in all manufacturing companies who want to have a sophisticated information system for production planning and scheduling. PPS standard provides XML schema and communication protocols for information exchange among manufacturing application programs in the web-services environment. This document especially focuses on transaction messages that represent domain information in accordance with the context of the communication, as well as transaction rules for contexts such as pushing and pulling of the information required.

Status:

This document was last revised or approved by the PPS TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/pps/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/pps/ipr.php>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/pps/>.

Notices

Copyright © OASIS® 2007. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", PPS are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1	Introduction.....	7
1.1	Terminology	7
1.2	Normative References	7
1.3	Non-Normative References	7
1.4	Conformance	8
1.5	Terms and definitions	8
2	Messaging model	9
2.1	Basic models	9
2.2	Message classes	9
2.3	Messaging models.....	10
2.3.1	Add transaction	10
2.3.2	Change transaction	10
2.3.3	Remove transaction	11
2.3.4	Notify transaction.....	11
2.3.5	Sync transaction.....	11
2.3.6	Get-Show transaction.....	11
2.4	Procedures on responders	12
2.4.1	Common tasks.....	12
2.4.2	Confirm message	12
2.4.3	Error handling.....	12
3	Message document	14
3.1	Message Structure.....	14
3.2	Transaction element	14
3.3	Multiple documents message	16
4	Add, Change and Remove transaction	17
4.1	Add transaction.....	17
4.2	Change transaction.....	18
4.2.1	Insert property	18
4.2.2	Update property.....	19
4.2.3	Delete property.....	19
4.3	Remove transaction.....	20
5	Notify and Sync Transactions.....	21
5.1	Notify transaction	21
5.2	Sync transaction	21
5.2.1	Sync message	22
5.2.2	Procedure of information owner	23
6	Information Query.....	24
6.1	Target domain objects	24
6.1.1	Selection by object IDs.....	24
6.1.2	Selection by Property elements.....	24
6.1.3	Disjunctive and conjunctive conditions.....	25
6.1.4	Selection by wildcard.....	26
6.2	Target domain property	26

6.2.1 All available properties	26
6.2.2 Selecting domain property.....	27
6.2.3 Sorting by property value	27
6.2.4 Calculation of property value.....	28
6.3 Multiple property	29
6.4 Using Header element.....	30
6.4.1 Query by header element.....	30
6.4.2 Count of domain objects.....	31
6.5 Show message	31
6.5.1 Structure of Show message	31
6.5.2 Header in Show message	31
7 XML Elements	33
7.1 Error element.....	33
7.2 App element.....	34
7.3 Condition element.....	34
7.4 Selection element	35
7.5 Header element	35
7.6 Property element	36
A. Implementation level.....	38
B. Acknowledgements	40
C. Revision History.....	41

Figures

Figure 1 Basic types of messaging	9
Figure 2 Add transaction.....	10
Figure 3 Change transaction	10
Figure 4 Remove transaction.....	11
Figure 5 Notify transaction	11
Figure 6 Sync transaction	11
Figure 7 Get-Show transaction	12
Figure 8 Add message transactions	17
Figure 9 Change message transactions	18
Figure 10 Remove message transactions	20
Figure 11 Notify message transactions	21
Figure 12 Sync message transaction	22
Figure 13 Get -Show message transactions.....	24
Figure 14 Single property and Multiple property.....	29

1 Introduction

This part of PPS standard provides specifications of XML transaction elements for messaging between two application programs. XML representations of the messages consist of XML core elements that are defined in [PPS01]. This part defines additional XML elements and attributes that are needed to establish such communications.

From perspective of planning and scheduling in manufacturing management, there are many kinds of domain documents and domain objects. All of that information are sent or received in particular context such as notifying new information, requesting particular information, and so forth. This part prescribes communication protocols by categorizing such various transactions into simple models. This standard doesn't focus on the underlying communication protocols, such as HTTP, SMTP and FTP. This standard allows all readers to select any low-level protocols to establish the communication properly in a secure way.

A transaction element corresponds to a message document which is sent or received as a message. This part does not define transaction element, but defines a data structure of transaction elements that may be created for any particular circumstances. Each transaction element has domain objects of production planning and scheduling. The domain objects are represented by nine primitive elements defined in [PPS01]. All domain objects defined in this standard are sub-classes of the primitive elements.

This part of the standard also defines messaging models of communication between two application programs, where a transaction element is sent as a message. In the messaging model, an initiator can invoke a service such as add, change and remove information of the responder. The initiator is also able to request of getting information by sending a query-like formatted message. This part of standard defines syntax and rules for such messaging models.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2 Normative References

- [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- [PPS01] PPS (Production Planning and Scheduling) Part 1: Core Elements, Version 1.0, Public Review Draft 01, <http://www.oasis-open.org/committees/pps/>
- [PPS03] PPS (Production Planning and Scheduling) Part 3: Profile Specifications, Version 1.0, Public Review Draft 01, <http://www.oasis-open.org/committees/pps/>

1.3 Non-Normative References

- [PSLXWP] PSLX Consortium, PSLX White Paper - APS Conceptual definition and implementation, <http://www.pslx.org/>
- [PSLX001] PSLX Technical Standard, Version 2, Part 1: Enterprise Model (in Japanese), Recommendation of PSLX Forum, <http://www.pslx.org/>
- [PSLX002] PSLX Technical Standard, Version 2, Part 2: Activity Model (in Japanese), Recommendation of PSLX Forum, <http://www.pslx.org/>
- [PSLX003] PSLX Technical Standard, Version 2, Part 3: Object Model (in Japanese), Recommendation of PSLX Forum, <http://www.pslx.org/>

44 **1.4 Conformance**

45 A document or message confirms OASIS PPS Transaction Messages if all elements in the artifact are
46 consistent with the normative text of this specification, and the document can be processed properly with
47 the XML schema that can be downloaded from the following URI.

48

49 <http://docs.oasis-open.org/ppsv1.0/ppsv1.0-transaction-messages.xsd>

50

51 A Process or service conforms OASIS PPS Transaction Messages if the process or service can deal with
52 the message that conforms OASIS PPS Transaction Messages and the process or service is consistent
53 with the normative text of this specification

54 **1.5 Terms and definitions**

55 **Messaging model**

56 Simple patterns of messaging between sender and receiver, or requester and responder. The six
57 message models are defined from an application independent perspective, by defining eight
58 different message types as components.

59 **Primitive element**

60 XML element that represents a primitive object in the production planning and scheduling domain.
61 Nine primitive elements are defined in [PPS01]. Every domain objects are represented by the
62 primitive elements.

63 **Transaction element**

64 XML element that represents a message document to be sent or received between application
65 programs. Transaction element has primitive elements to represent any objects of domain
66 information. Transaction element may have a header information and application specific
67 information.

68 **Domain document**

69 Document that is the content of message sent or received between application programs.
70 Message document consists of a verb part and a noun part. Verbs such as add, change and
71 remove affect the types of messages, while nouns show the classes of domain objects.

72 **Domain object**

73 Object that corresponds to production planning and scheduling information in manufacturing
74 operations management. Domain objects are contents of transaction elements, and represented
75 by primitive elements.

76 **Domain property**

77 Any parameters that show a property of a domain object. A domain property is represented by
78 XML attributes of the primitive element, or XML child elements of the primitive elements. A
79 domain object may have multiple domain properties that has same property name.

80 **Implementation profile**

81 Specification of capability of an application program. The profile includes a list of available
82 documents and transactions that may be exchanged in PPS transaction messages among
83 production planning and scheduling applicaions.

84 **Application profile definition**

85 Collections of profile specifications for all application programs that may be involved in the
86 communication goup who exchanges PPS transaction messages. This provides all available
87 domain documents, domain objects and domain peoperties.

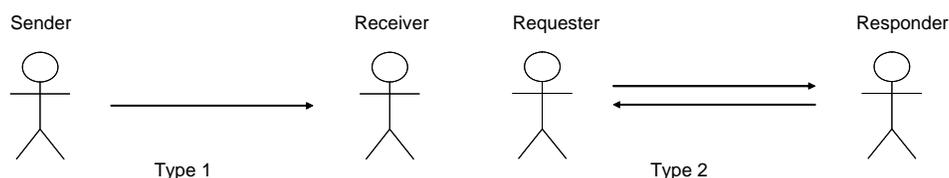
88

89 2 Messaging model

90 2.1 Basic models

91 Two basic types of messaging are defined in this part of PPS standard. The first one is a communication
92 between sender and receiver (Type 1), where the sender simply sends a message to the receiver without
93 any negotiations. The second is a communication between requester and responder (Type 2), where the
94 requester asks the responder to do some services. The responder may answer to the sender by sending
95 appropriate message. The receiver or responder may be multiple at one transaction, so as to make broad
96 casting.

97



98

99

Figure 1 Basic types of messaging

100

101 In many practical business situations, communication protocols such as customer negotiation with price
102 and due dates, communication procedures are designed using these basic patterns as building blocks. In
103 such cases, how to combine the component is not in the scope of this standard.

104 In addition, underlying communication protocols such as HTTP and TCP/IP may used to define for these
105 simple patterns, considering security, reliability, efficiency and so forth. In such cases, messages may be
106 sent several times for one arrow in Figure 1. This is also not in the scope of this part.

107 2.2 Message classes

108 Message documents, which represent message between sender and receiver, or requester and
109 responder, are defined for each messaging transaction. According to the verb information of each
110 message document, they can be categorized into 8 different classes. The table shows the message types.

111

112 Table 1 Message type in transaction models

Message classes	Description
Add	The message requests to add the specified domain objects to the database managed by the responder.
Change	The message requests to change the specified domain objects in the database managed by the responder.
Remove	The message requests to remove the specified domain objects in the database managed by the responder.
Confirm	The message is sent to the requester of request from the responder as a confirmation of the results.
Notify	The message is sent any domain objects to the receiver as information notification from the sender.
Sync	The message requests the owner of information to send notify message

	synchronously at the time the specified event occurs.
Get	The message asks the responder to show the specified domain objects in a specified format by sending Show message.
Show	The message has the requested information of domain objects to answer to the Get message from the responder.

113

114 In order to ask the confirmation from responders, message documents of Add, Change, Remove and
 115 Sync MAY have an attribute of the following confirmation requests.

116

117 *Table 2 Confirmation request*

Confirm type	Description
Never	Responder SHOULD NOT respond to the request.
OnError	Responder SHOULD respond to the request, only if any errors in processing the request occur.
Always	Responder SHOULD always respond to the request.

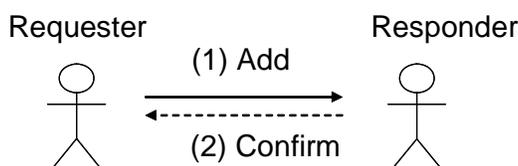
118

119 2.3 Messaging models

120 2.3.1 Add transaction

121 In Add transaction model, the requester sends an Add message to request responder to add the specified
 122 domain objects to the database that is managed by the responder. After making the task of adding the
 123 information, the responder can send a Confirm message depending on the confirmation request.

124



125

126

Figure 2 Add transaction

127

128 2.3.2 Change transaction

129 Change model performs when the requester tries to change the specified domain objects in the database
 130 that is managed by the responder. The requester sends a Change message to the responder as a
 131 request to change. The responder can do the task and send a Confirm message as a result of the task.

132



133

134

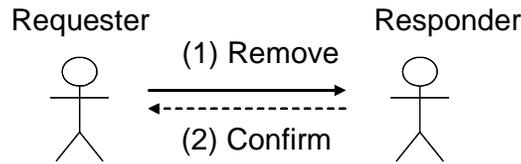
Figure 3 Change transaction

135

136 2.3.3 Remove transaction

137 Remove model performs when the requester tries to delete the specified domain objects in the database
138 managed by the responder. The requester sends a Remove message, and the responder responds a
139 Confirm message if the Remove message has a confirmation request.

140



141

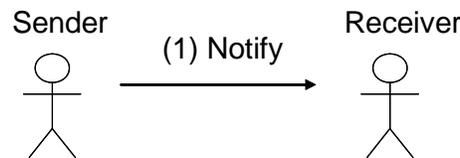
142

Figure 4 Remove transaction

143 2.3.4 Notify transaction

144 Basic pattern 1 performs in the Notify model. In this model, the sender sends a Notify message to the
145 receiver. There is no obligation on the receiver to respond to the message nor make a task for the
146 message.

147



148

149

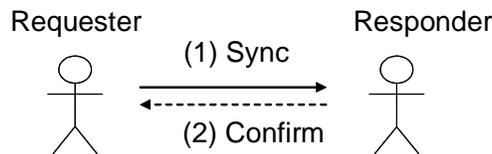
Figure 5 Notify transaction

150

151 2.3.5 Sync transaction

152 Sync transaction performs that requester requests responder to send Notify message synchronously at
153 the time when the specified event occurs on the domain objects owned by the responder. Responder
154 keeps monitoring the event to send Notify messages by invoking another Notify transaction.

155



156

157

Figure 6 Sync transaction

158 2.3.6 Get-Show transaction

159 Get-Show transaction performs like a query-response process in the client-server database systems. The
160 requester sends a Get message to the responder in order to get information of the specified domain
161 objects. The responder tries to answer the request by sending Show message with corresponding
162 information that is managed.

163

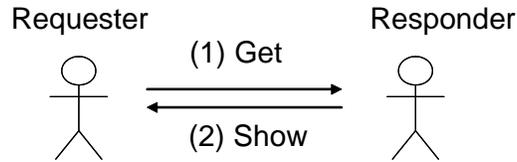


Figure 7 Get-Show transaction

164
165
166
167

168 2.4 Procedures on responders

169 2.4.1 Common tasks

170 Responders SHOULD have the capability to perform the following tasks when a message document is
171 received.

- 172 ● The responder, who receives a proper Get message, SHOULD send a Show message to the
173 requester. The Show message SHOULD have either error information or domain object requested
174 by the requester in the specified forms.
- 175 ● The responder, who receives a proper Add message, SHOULD add the domain objects in the
176 message to the database that is managed by the responder, unless the ID of the object already
177 exists.
- 178 ● The responder, who receives a proper Change message, SHOULD change the target domain
179 object in the database managed by the responder to the new data in the message, unless the ID of
180 the object doesn't exist.
- 181 ● The responder, who receives a proper Remove message, SHOULD delete the target domain
182 object in the database managed by the responder, unless the ID of the object doesn't exist.

183 2.4.2 Confirm message

184 The responder of Add, Change, Remove and Sync message SHOULD have capability to make the
185 following tasks when the message received has a confirmation request.

- 186 ● The responder SHOULD send a Confirm message to the requester when the Add message
187 received has an attribute of confirm="Always". The Confirm message SHOULD have either error
188 information or the id list that shows all the objects added to the database.
- 189 ● The responder SHOULD send a Confirm message to the requester when the Change message
190 received has an attribute of confirm="Always". The Confirm message SHOULD have either error
191 information or the id list that shows all the objects changed in the database.
- 192 ● The responder SHOULD send a Confirm message to the requester when the Remove message
193 received has an attribute of confirm="Always". The Confirm message SHOULD have either error
194 information or the id list that shows all the objects deleted in the database.
- 195 ● The responder SHOULD send a Confirm message to the requester when the Sync message
196 received has an attribute of confirm="Always". The Confirm message SHOULD have either error
197 information or the id list that shows all the objects to be monitored for synchronization.

198 2.4.3 Error handling

199 To deal with errors occurred during the process of messaging, e.g. syntax or semantic problems detected
200 by the responder's programs, the responder SHOULD have a capability of the following error handling:

- 201 ● The responder, who receives a Get message and is hard to respond in normal processes because
202 of errors, SHOULD send a Show message with the error information to the requester.

- 203 ● The responder who receives an Add message with the attribute of confirm="OnError" and is hard
204 to respond in normal processes because of errors, SHOULD send a Confirm message with the
205 error information to the requester.
- 206 ● The responder who receives a Change message with the attribute of confirm="OnError" and is
207 hard to respond in normal processes because of errors, SHOULD send a Confirm message with
208 the error information to the requester.
- 209 ● The responder who receives a Remove message with the attribute of confirm="OnError" and is
210 hard to respond in normal processes because of errors, SHOULD send a Confirm message with
211 the error information to the requester.
- 212 ● The responder who receives a Sync message with the attribute of confirm="OnError" and is hard to
213 respond in normal processes because of errors, SHOULD send a Confirm message with the error
214 information to the requester.
- 215

216

3 Message document

217

3.1 Message Structure

218 A message that is exchanged between two parties SHOULD consist of one or more message documents.
219 When more than two message documents are sent in one message, one of those documents SHOULD
220 be assigned as a primary message document.

221 A message that is not a primary message document SHOULD have relation to a primary message
222 document or other message documents that is in the same message. This recursive relation SHOULD
223 show the primary message document at the end of the linkage.

224 Since this standard doesn't address on how to exchange messages in IP (Internet Protocol) level, data
225 envelope mechanisms such as SOAP can be considered as well as a simple SMTP and file transfer
226 mechanism.

227

3.2 Transaction element

228 A domain document is represented by a transaction element. This section defines the common data
229 structure of the transaction elements. The list of the transaction elements which are necessary for
230 production planning and scheduling are address in PPS standard profile.

231 The structure of the transaction element SHOULD be consistent with the following XML schema and the
232 specifications.

233

```
234 <xsd:complexType name="TransactionType">  
235   <xsd:sequence>  
236     <xsd:element ref="Error" minOccurs="0" maxOccurs="unbounded"/>  
237     <xsd:element ref="App" minOccurs="0"/>  
238     <xsd:element ref="Spec" minOccurs="0" maxOccurs="unbounded"/>  
239     <xsd:element ref="Condition" minOccurs="0" maxOccurs="unbounded"/>  
240     <xsd:element ref="Selection" minOccurs="0" maxOccurs="unbounded"/>  
241     <xsd:element ref="Header" minOccurs="0"/>  
242     <xsd:choice minOccurs="0">  
243       <xsd:element ref="Party" minOccurs="0" maxOccurs="unbounded"/>  
244       <xsd:element ref="Plan" minOccurs="0" maxOccurs="unbounded"/>  
245       <xsd:element ref="Order" minOccurs="0" maxOccurs="unbounded"/>  
246       <xsd:element ref="Item" minOccurs="0" maxOccurs="unbounded"/>  
247       <xsd:element ref="Resource" minOccurs="0" maxOccurs="unbounded"/>  
248       <xsd:element ref="Process" minOccurs="0" maxOccurs="unbounded"/>  
249       <xsd:element ref="Lot" minOccurs="0" maxOccurs="unbounded"/>  
250       <xsd:element ref="Task" minOccurs="0" maxOccurs="unbounded"/>  
251       <xsd:element ref="Operation" minOccurs="0" maxOccurs="unbounded"/>  
252     </xsd:choice>  
253   </xsd:sequence>  
254   <xsd:attribute name="id" type="xsd:string" use="required"/>  
255   <xsd:attribute name="ref" type="xsd:string"/>  
256   <xsd:attribute name="action" type="xsd:string"/>  
257   <xsd:attribute name="option" type="xsd:string"/>  
258   <xsd:attribute name="transaction" type="xsd:string"/>  
259   <xsd:attribute name="confirm" type="xsd:string"/>  
260   <xsd:attribute name="profile" type="xsd:string"/>  
261   <xsd:attribute name="sender" type="xsd:string"/>  
262   <xsd:attribute name="create" type="xsd:dateTime"/>  
263   <xsd:attribute name="description" type="xsd:string"/>  
264 </xsd:complexType>
```

265

- 266 ● *id* attribute SHOULD represent the identifier of the message. Every transaction message SHOULD
267 have a unique id in the scope of the sender or the requester.

- 268 ● *ref* attribute SHOULD represent the identifier of a primary message document or other message
269 document that is in the same message, when the message has more than one message document.
- 270 ● *action* attribute SHOULD represent the type of the message, where the types correspond to verbs
271 information for the message. Values of the attribute SHOULD be either "Add", "Change", "Remove",
272 "Confirm", "Notify", "Sync", "Get", or "Show".
- 273 ● *option* attribute SHOULD represent any optional information that may be interpreted by the
274 receiver of the message.
- 275 ● *transaction* attribute SHOULD represent the identifier of the series of a transaction. If the message
276 has a predecessor that relates to the message, the values of this attribute in those messages
277 SHOULD be the same. For example, messages in the same messaging model SHOULD have the
278 same value on the transaction attribute. Re-sending messages SHOULD have the same value of
279 transaction attribute with the original message.
- 280 ● *confirm* attribute SHOULD represent a confirmation request. The value of the attribute MSUT be
281 either "Never", "OnError", or "Always".
- 282 ● *profile* attribute SHOULD identify the application profile data, which describes details of domain
283 objects and domain properties to adjust the particular business applications. The empty value of
284 the attribute SHOULD represent the transaction follows the standard profile.
- 285 ● *sender* attribute SHOULD represent an identifier of the sender or requester of the message. This
286 information is not for the low-level communication programs but for application programs.
- 287 ● *create* attribute SHOULD represent a date when the transaction document is created.
- 288 ● *description* attribute SHOULD represent any comments or descriptions.

289

290 Elements under the transaction element SHOULD follow the sentences:

- 291 ● *Error* element SHOULD represent error information.
- 292 ● *App* element SHOULD represent any information for the application programs.
- 293 ● *Spec* element SHOULD represent any particular specification of the transaction.
- 294 ● *Condition* element SHOULD represent any condition of selecting required domain objects.
- 295 ● *Selection* element SHOULD represent any condition of selecting required properties of a domain
296 object.
- 297 ● *Header* element SHOULD represent information for detailed messaging described in the section 6.
- 298 ● *Party, Plan, Order, Item, Resource, Process, Lot, Task, or Operation* element SHOULD represent
299 domain objects. Different type of them SHOULD NOT be specified at the same transaction element.

300

301 A message type that the transaction element is addressed determines the combination of elements
302 available to specify. The table below shows the combination matrix. Each column shows different
303 message transaction type, while the row shows available elements in the transaction element. The blank
304 cell represents the corresponding element SHOULD NOT be the child of the transaction element. "M"
305 denotes that the corresponding element SHOULD be defined in the parent element. And "O" denotes
306 optional where the element may be described depending on the situation.

307

308

309 *Table 3 Structure of transaction element*

	Add	Change	Remove	Confirm	Confirm (Error)	Notify	Sync	Get	Show	Show (Error)
<i>Error element</i>					M					M
<i>App element</i>	O	O	O	O	O	O	O	O	O	O
<i>Condition element</i>	O	O	O				O	O		
<i>Selection element</i>		M						O		
<i>Header element</i>						M		O	M	O
<i>Primitive element</i>	M			M		M			M	

310 **3.3 Multiple documents message**

311 A message MAY consist of more than one domain documents. When one document is in a message, the
 312 type of message is the same as the type of document. When more than one documents are in the
 313 message, first, the primary message document is the same as the type of message.

314 For other documents in multiple documents message, the available combination of document types and
 315 message types SHOULD be in the matrix of the Table below. In the table, “Yes” denotes the combination
 316 is available.

317

318 *Table 4 Available combination of multiple documents*

Primary Document	Subsidiary document							
	Add	Change	Remove	Confirm	Notify	Sync	Get	Show
Add	Yes				Yes			
Change		Yes			Yes			
Remove			Yes		Yes			
Confirm				Yes				
Notify					Yes			
Sync								
Get					Yes		Yes	
Show					Yes			Yes

319

320

4 Add, Change and Remove transaction

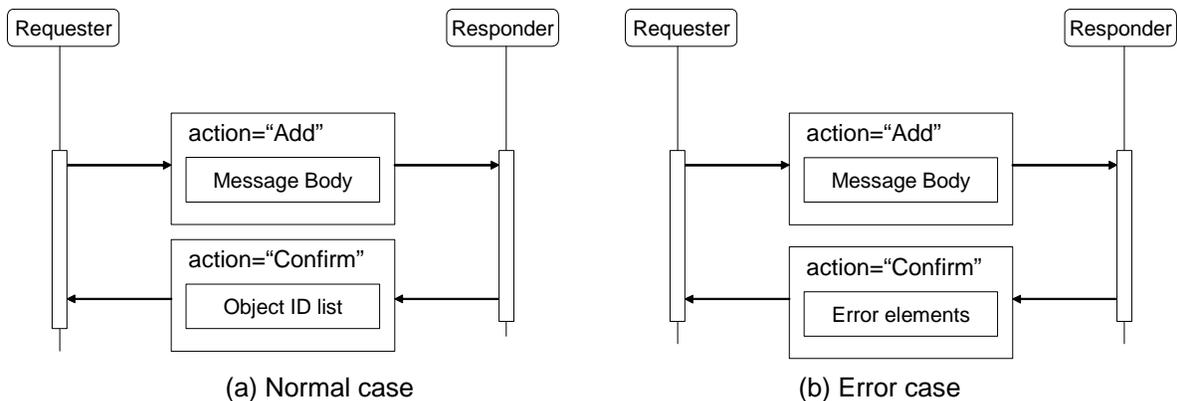
4.1 Add transaction

322 Add message requests the responder to add the specified domain objects in the message to the
323 database managed by the responder.

324 If the Add message request to add domain objects with ID specified at the "id" attribute, responder
325 SHOULD check existence of the ID in its database and reject the request when the corresponding data
326 already exists in the database. If the message has an ID that already exists in the database, the
327 responder SHOULD NOT add the data.

328 If the Add message request to add domain object without ID, the responder SHOULD create any unique
329 ID in its database, and create a new domain object that has the specified information. The new IDs MAY
330 return by Confirm message if the requester needs confirmation.

331



332

333

334

Figure 8 Add message transactions

335

336 Example 1-A: Message to add three Product Records

337

338

339

340

341

```

<ProductRecord id="A-1" transaction="01" action="Add" sender="A">
<Item id="001" name="Product-1"><Spec type="pps:color"><Char value="red"/></Spec></Item>
<Item id="002" name="Product-2"><Spec type="pps:color"><Char value="red"/></Spec></Item>
<Item id="003" name="Product-3"><Spec type="pps:color"><Char value="red"/></Spec></Item>
</ProductRecord>

```

342

343 When *Condition* element is specified in a transaction element, the *Property* element in the *Condition*
344 element shows common property of domain objects listed in the document. The following example is the
345 same request compare to the previous example.

346

347 Example 1-B: Add message using a *Condition* element

348

349

350

351

352

353

354

355

```

<ProductRecord id="A-2" transaction="02" action="Add" sender="A">
<Condition>
<Property name="pps:color"><Char value="red"/></Property>
</Condition>
<Item id="001" name="Product-1"/>
<Item id="002" name="Product-2"/>
<Item id="003" name="Product-3"/>
</ProductRecord>

```

356

357 The response to Add message is done by sending a Confirm message that has primitive elements in its
 358 body. The primitive element represents the domain object that is successfully added, and SHOULD only
 359 have *id* attribute. The next example is the Confirm message as a result of the previous Add message.

360

361 Example 1-C: Confirm message as a response of an Add transaction

```
362 <ProductRecord id="B-1" transaction="01" action="Confirm" sender="B">
363 <Item id="001" />
364 <Item id="002" />
365 <Item id="003" />
366 </ProductRecord>
```

367

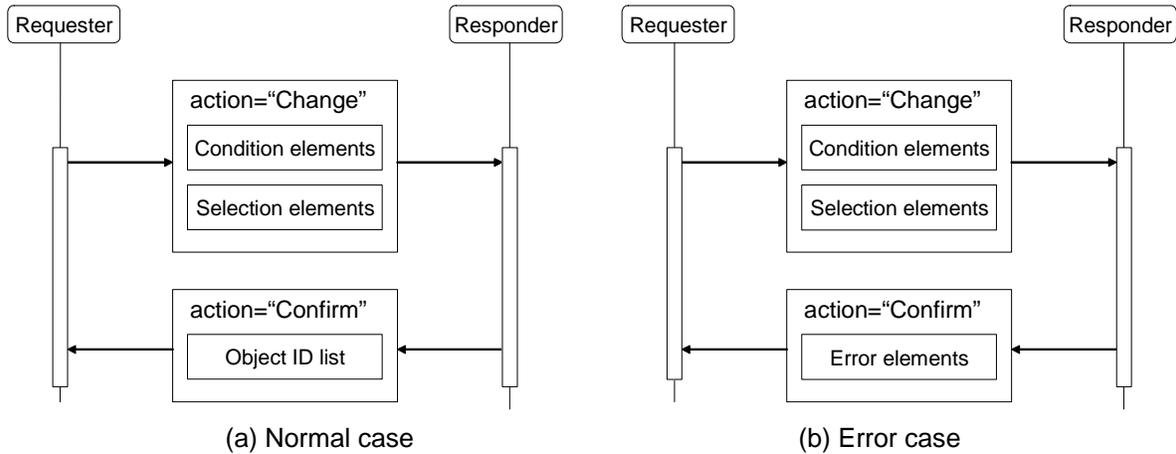
368 4.2 Change transaction

369 Change message requests to change the specified information of the specified domain objects that is in
 370 the database managed by the responder. In order to identify the target domain object, *Condition* element
 371 has any condition to select one of more domain objects.

372 After selecting the target domain object, *Select* element SHOULD represent the values of target
 373 properties to be changed. The values SHOULD be specified in the *Property* element in the *Selection*
 374 element.

375 All the selected domain objects depending on the *Condition* element SHOULD be applied to change in
 376 the same way. ID of domain objects SHOULD NOT be changed by Change transactions.

377



378

379

380

Figure 9 Change message transactions

381

382 In the database managed by the responder, a property type is either single or multiple. If the property
 383 type is single, the value requested to change is applied as a new value of the property. Otherwise, in the
 384 cases of multiple properties, the property of the domain object is inserted, updated or deleted depending
 385 on the type of the Change message.

386 4.2.1 Insert property

387 For the multiple primitives that can be duplicated in the same object, an insert property message performs
 388 to add another property that has a new value. When *type* attribute of *Selection* element has "insert" value,
 389 it shows that the properties in the *Selection* element are requested to insert.

390

391 Example 2: Add information of new level 10 as the latest stock value.

```
392 <ProductRecord id="A-4" transaction="04" action="Change" sender="A">
```

```
393 <Condition id="001"/>
394 <Selection type="insert" >
395 <Property name="pps:stock"><Qty value="10"/></Property>
396 </Selection>
397 </ProductRecord>
```

398

399 4.2.2 Update property

400 When the value of *type* attribute of *Selection* element is "update", the properties in the *Selection* element
401 are for updating the current properties. The target properties to be changed are selected by *Condition*
402 elements which are defined under the *Selection* element.

403 If the *Condition* elements select more than one property instances, all values of these instances are
404 changed to the value specified in the *Property* element. If the *Condition* elements select no property
405 instance, nothing happens for the message.

406

407 Example 3-A: The message requests to change the usage of A001-2 from 1 to 4.

```
408 <ProductRecord id="A-5" transaction="05" action="Change" sender="A">
409 <Condition id="A001"/>
410 <Selection type="update" >
411 <Condition><Property name="pps:child"><Char value="A001-2"/></Property></Condition>
412 <Property name="pps:child-value"><Qty value="4"/></Property>
413 </Selection>
414 </ProductRecord>
```

415

416 Example 3-B: Initial status of the product data A001 that has A001-1, A001-2 and A001-3.

```
417 <Item id="A001">
418 <Compose type="pps:child" item="A001-1"><Qty value="1"/></Compose>
419 <Compose type="pps:child" item="A001-2"><Qty value="1"/></Compose>
420 <Compose type="pps:child" item="A001-3"><Qty value="1"/></Compose>
421 </Item>
```

422

423 Example 3-C: Revised status of the product data

```
424 <Item id="A001">
425 <Compose type="pps:child" item="A001-1"><Qty value="1"/></Compose>
426 <Compose type="pps:child" item="A001-2"><Qty value="4"/></Compose>
427 <Compose type="pps:child" item="A001-3"><Qty value="1"/></Compose>
428 </Item>
```

429

430 4.2.3 Delete property

431 When a value of *type* attribute of *Selection* element is "delete", then it performs to delete particular
432 properties that are selected by *Condition* elements under the *Selection* element. *Condition* element is
433 necessary to select the target properties to be deleted.

434 If the *Condition* elements select more than one property instances, all of these instances are deleted. If
435 the *Condition* elements select no property instance, nothing happens for the message.

436

437 Example 4: Usage of "Machine-1" by the process "Proc-1" is requested to delete.

```
438 <ProcessRecord id="A-6" transaction="06" action="Change" sender="A">
439 <Condition id="Proc-01"/>
440 <Select type="delete">
441 <Condition><Property name="pps:equipment"><Char value="Machine-1"/></Property></Condition>
442 </Selection>
```

443

```
</ProcessRecord>
```

444

445 Example 5: Delete all inventory records of the item "A001" that has a date before August 1st.

446

```
<InventoryRecord id="A-7" transaction="07" action="Change" sender="A">
447 <Condition id="A001"/>
448 <Selection type="delete">
449 <Condition><Property name="pps:stock-date">
450 <Time value="2006-08-01T00:00:00" condition="latest"/></Property>
451 </Condition>
452 </Selection>
453 </InventoryRecord>
```

454

455 4.3 Remove transaction

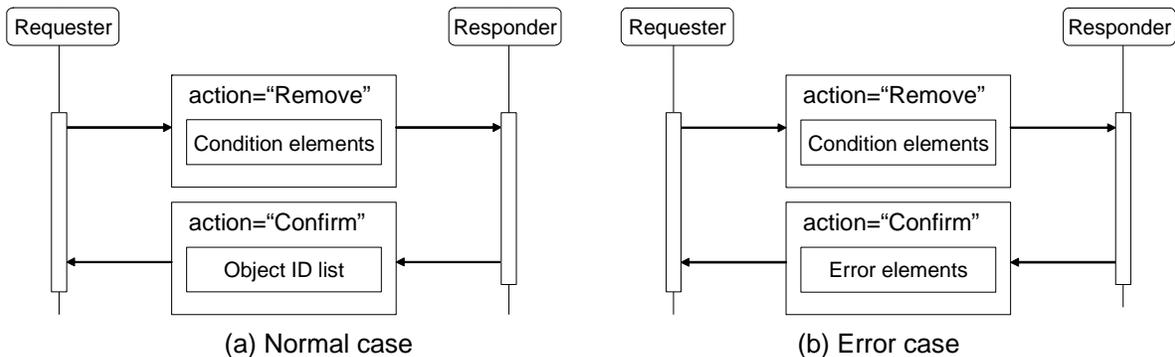
456

457 Remove message requests to delete the specified domain objects in the database managed by the responder. The responder can decide either the request is accepted or rejected. If it is rejected, the responder SHOULD send error message, unless the confirm attribute is "Never". Removing objects means that the data in the database is actually deleted, or logically deleted such that only the delete flag is marked on the object.

458

459 The target domain objects to be removed are selected by specifying *Condition* elements that represent the conditions of target domain objects.

460



464

465

(a) Normal case

(b) Error case

Figure 10 Remove message transactions

466

467

468

Example 6: A message requesting that all the lot schedule objects of item "M001" are removed.

469

```
<LotSchedule id="A-8" transaction="08" action="Remove" sender="A">
470 <Condition>
471 <Property name="pps:item"><Char value="M001"/></Property>
472 </Condition>
473 </LotSchedule>
```

474

475

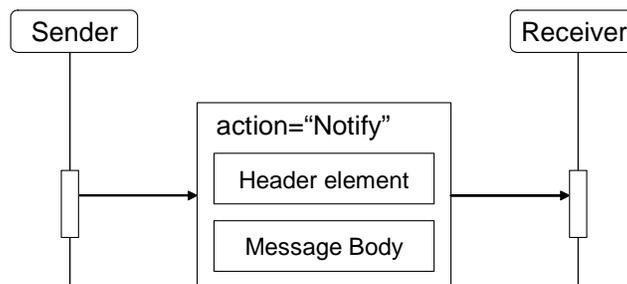
476 5 Notify and Sync Transactions

477 5.1 Notify transaction

478 Notify message SHOULD have "Notify" in the *action* attribute. The figure shows that transaction pattern of
479 Notify message exchange. Notify message doesn't need its response.

480 Notify message MAY be sent by the sender to any information users that the sender decides as the
481 destination of the message. If Notify message is caused by synchronization request defined by a Sync
482 message received in advance, the message is sent when the corresponding event occurs. In Notify
483 message for synchronization, the *event* attribute SHOULD show the event name.

484



485

486 *Figure 11 Notify message transactions*

487

488 Notify message SHOULD have a *Header* element that MAY have the number of domain objects and any
489 aggregated information of objects. Domain objects, which are represented by primitive elements
490 described in [PPS01], MAY be specified in the body of Notify message.

491

492 Example 7: A Notify message shows reception of customer order 001 and its detailed items.

493

```
494 <CustomerOrder id="A-9" transaction="09" action="Notify" sender="A">  
495 <Header id="001" count="3" title="Order Form">  
496 <Property type="target" name="pps:party" display="C-Name"><Char value="K-Inc."/></Property>  
497 <Property type="selection" name="pps:id" display="P/N"/>  
498 <Property type="selection" name="pps:name" display="NAME"/>  
499 <Property type="selection" name="pps:qty" display="QTY"/>  
500 <Property type="selection" calc="sum" name="pps:price" display="PRICE"><Qty value="1200"/></Property>  
501 </Header>  
502 <Order id="001-1" item="Product-A1"><Qty value="1"/></Order>  
503 <Order id="001-2" item="Product-A2"><Qty value="10"/></Order>  
504 <Order id="001-3" item="Product-A3"><Qty value="3"/></Order>  
</CustomerOrder>
```

505

506 5.2 Sync transaction

507 In order to synchronize information of users with the information owner, the user needs to know the
508 change of information at the time it occurs. The sync transaction allows the user to request the
509 information owner to notify the change of domain objects synchronously.

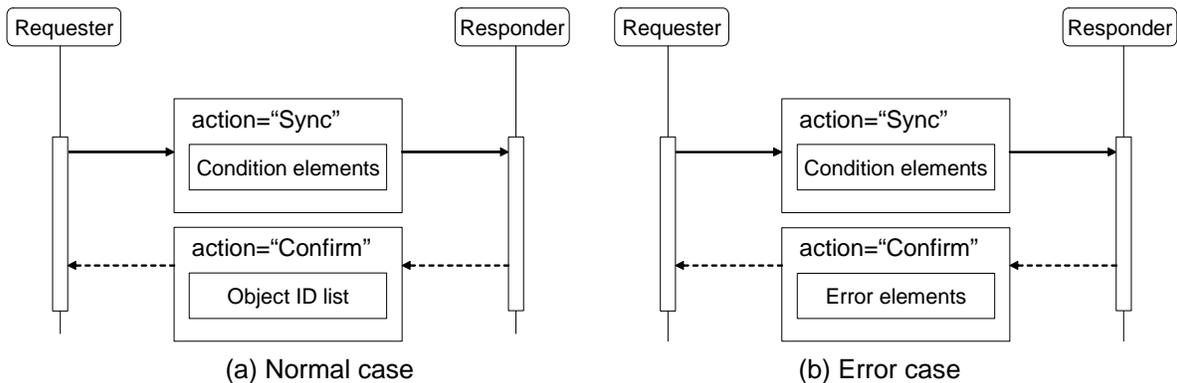
510 If an information owner monitors particular property value of a domain object and tries to detect certain
511 event occurrence such as data changes, the sync message is used to establish synchronization by
512 requesting subscription of the event occurrence detected by the information owner.

513 When a synchronization request by sync a message is accepted by responder, e.g., the information
514 owner, the responder SHOULD send a notification message by invoking another transaction when the

515 corresponding event occurs. The notification messages are not included in this Sync transaction.
 516 Notification of change of the property value will be invoked as a different transaction independent from the
 517 sync transaction.

518 This transaction is similar to publish-subscription model. The sync message can be regarded as a
 519 subscription request message. If the responder has a subscription management module, then an
 520 application program sent a single Notify message to the module, which knows the subscribers and
 521 dispatch the message to all the members listed as a subscriber.

522



523

524

525

Figure 12 Sync message transaction

526

527 All properties of a domain object MAY NOT be available to provide this capability. In order to know the
 528 capability of application program and the name list of events that the application program can provide, an
 529 implementation profile defined in [PPS03] SHOULD specify the information.

530 According to the implementation profile of the responder, the responder (information owner) determines
 531 the interval of monitoring cycle, size of difference to detect changes, range of value to detect event
 532 occurrence by minimum and maximum constraints, and so forth [PPS03].

533 When the value of the property is changed into the range defined by maximum and minimum constraints,
 534 the information owner SHOULD send the notification. The owner SHOULD NOT send a next notification
 535 of the event unless the value will once be outside of the range.

536 When the size of difference to detect changes is defined, any changes of the property value less than the
 537 size SHOULD be ignored.

538 The changes during the monitoring cycle MAY be merged at the time of the next monitoring time.
 539 Therefore, changes during the cycle MAY NOT be detected by the requester.

540 5.2.1 Sync message

541 Sync message is a message to request synchronization of information by requester. Sync message
 542 SHOULD specify a value "Sync" at *action* attribute of the transaction element. Sync message SHOULD
 543 have an event name that has been defined in advance by the responder.

544 Sync message MAY specify particular domain objects that the responder needs to add the requester as a
 545 subscriber of the objects. *Condition* element allows the requester to make request of synchronization for
 546 several domain objects by sending one sync message.

547 When there is no available event specified by the event attribute in the suggested domain object, the
 548 responder SHOULD send a error information in Confirm message unless the request has "Never" value
 549 on the *confirm* attribute.

550

551 Example 8-A: To request notification when event "E01" occurs on any production order of item "A001".

552

553

554

```
<ProductionOrder id="A-3" transaction="03" action="Sync" event="E01" sender="A" confirm="Always">
  <Condition>
    <Property name="pps.item"><Char vaue="A001"/></Property>
```

```
555 </Condition>
556 </ProductionOrder>
```

557

558 Example 8-B: The requester is registered in the subscription list of event "E01" on the three orders.

```
559 <ProductionOrder id="B-1" transaction="03" action="Confirm" event="E01" sender="B">
560 <Order id="1201"/>
561 <Order id="1204"/>
562 <Order id="1223"/>
563 </ProductionOrder>
```

564

565 Once a Sync message is received without error, the request will be effective until the responder will get a
566 cancel request of the subscription, or the responder will stop the event detection. In order to cancel the
567 Sync request by requester, the requester SHOULD send a Sync message that has no value of *event*
568 attribute. When the responder receives a Sync message with no event name, it SHOULD cancel the
569 synchronization request sent by the requester for all domain objects specified in the message. The name
570 of requester is removed from the list of the subscription associated with the specified domain objects.

571 5.2.2 Procedure of information owner

572 Information owner, who has a capability of event publishing services, MAY specify the available event
573 information on the implementation profile described in [PPS03]. In accordance with the specification of
574 the profile, the owner SHOULD perform event detection and publication.

575 First, the information owner SHOULD monitor the actual value of the property that the owner decides to
576 detect the event. In every monitoring cycle, the owner SHOULD determine whether the event occurs, that
577 is, the value of the data is changed to satisfy all the conditions defined to the event. The conditions
578 include minimum value, maximum value, and difference of change of the domain property.

579 When the event occurs, the information owner SHOULD send a Notify message to all the members who
580 are in the list of subscription. This is similar to publish-subscription mechanism, so the information owner
581 MAY ask the publication to a middle-ware information broker.

582 The Notify message SHOULD have the event name at the *event* attribute. The transaction id SHOULD be
583 deferent from the transaction id of the corresponding Sync message. The Notify message of this event
584 occurrence SHOULD have the id of the domain object and the value of the property in the message body.

585

586 Example 8-C: Notify of event "E01" that shows a change of "production result" of production orders.

```
587 <ProductionOrder id="B-2" transaction="04" action="Notify" event="E01" sender="B">
588 <Order id="1204">
589 <Produce><Qty value="200"/></Produce>
590 </Order>
591 </ProductionRecord>
```

592

593

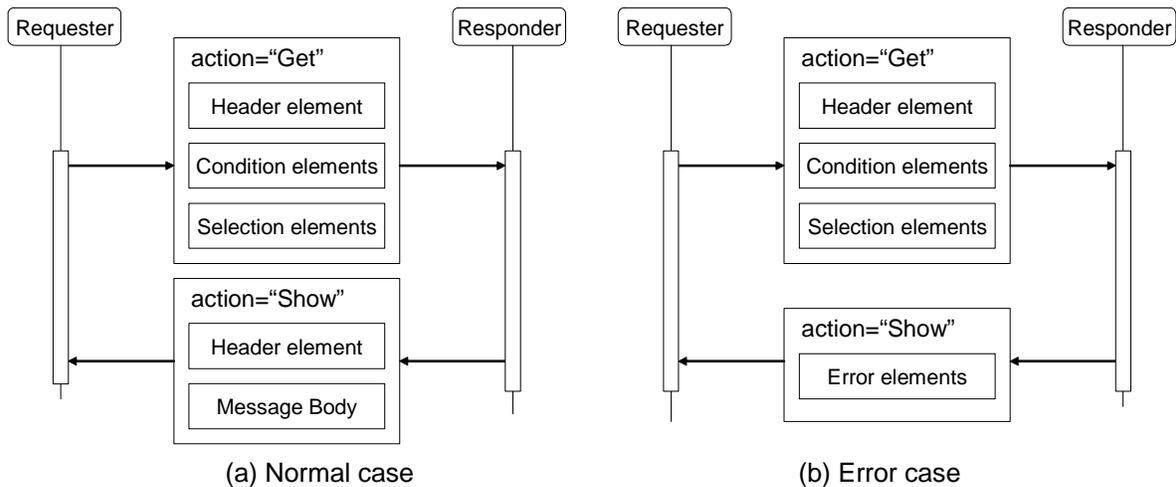
6 Information Query

594 Using a Get message, the requester MAY request particular information to the responder by specifying
595 the *Condition* elements that can select the target domain objects. The target objects can be specified
596 directly by IDs in *id* attribute, or any conditions of the domain objects using *Selection* elements.

597 If no *Condition* element is specified in Get message, all domain objects that the responder manages in
598 the database SHOULD be selected as the content of the Show message.

599 The responder who receives the Get message SHOULD process either by responding corresponding
600 domain objects, or refusing the request and setting error message in the Show message.

601



602

603

604

605

Figure 13 Get -Show message transactions

6.1 Target domain objects

6.1.1 Selection by object IDs

608 The easiest way to select domain objects is specifying IDs of the target objects in *Condition* elements. If
609 the ID of the object is known, it can be specified as a value of *id* attribute of a *Condition* element. In this
610 case, the *Condition* elements SHOULD be specified as many as the number of requested objects.

611

612 Example 9: Three objects that have "0001", "0005", "0013" as ID are requested.

```

613 <PartyRecord id="A-2" transaction="02" action="Get" sender="A">
614 <Condition id="0001"/>
615 <Condition id="0005"/>
616 <Condition id="0013"/>
617 <Selection type="all"/>
618 </PartyRecord>

```

619

6.1.2 Selection by Property elements

621 The second way to select domain objects is to specify *Property* elements in the *Condition* element. The
622 *Property* elements in this case represent condition of domain objects that SHOULD have the
623 corresponding property. Each *Property* element shows the property name and its value, or range of value.

624 If the data type of value is string, then the property shows that the *value* attribute should have the
625 specified value.

626 In order to select domain objects, the responder SHOULD evaluate the truth of the constraint described in
627 the property, and if all the *Property* elements in the parent *Condition* element are satisfied, the domain
628 object SHOULD be selected.

629

630 Example 10: Products that have “white” as a value of color property are required.

```
631 <ProductRecord id="A-3" transaction="03" action="Get" sender="A">  
632 <Condition>  
633 <Property name="pps:color"><Char value="white" /></Property>  
634 </Condition>  
635 <Selection type="all"/>  
636 </ProductRecord>
```

637

638 When a property specified in the *Condition* element is multiple, that is, the property can have many
639 instances, the value of the corresponding *Property* element SHOULD meet at least one instance in the
640 multiple property values.

641

642 Example 11: Any product items that has “A001” item in its parts list is required.

```
643 <ProductRecord id="A-4" transaction="04" action="Get" sender="A">  
644 <Condition>  
645 <Property name="pps:child"><Char value="A001"/></Property>  
646 </Condition>  
647 <Selection type="all"/>  
648 </ProductRecord>
```

649

650 In order to select target objects, *Condition* element allows the requester to specify any range of property
651 value. The range can be specified in *Property* element using *Qty*, *Char*, *Duration*, and *Time* element that
652 has *condition* attribute. If *exclusive* attribute of the element is specified to “true”, the value of the
653 numerical property is exclusively applied in the constraint.

654

655 Example 12: The message requests any products that the price is \$2,000 or higher.

```
656 <ProductRecord id="A-5" transaction="05" action="Get" sender="A">  
657 <Condition>  
658 <Property name="pps:price"><Qty value="2000" condition="min"/></Property>  
659 </Condition>  
660 <Selection type="all"/>  
661 </ProductRecord>
```

662

663 6.1.3 Disjunctive and conjunctive conditions

664 When more than one *Property* elements are specified in a *Condition* element, it means that all conditions
665 represented by the *Property* elements SHOULD be satisfied.

666

667 Example 13: Both A001 and A002 are the child items of the product.

```
668 <ProductRecord id="A-6" transaction="06" action="Get" sender="A">  
669 <Condition>  
670 <Property name="pps:child"><Char value="A001"/></Property>  
671 <Property name="pps:child"><Char value="A002"/></Property>  
672 </Condition>  
673 <Selection type="all"/>  
674 </ProductRecord>
```

675

676 When there are more than one *Condition* elements in a transaction document, these conditions are
677 interpreted disjunctive, i.e., at least one condition SHOULD be satisfied.

678

679 Example 14: Compare to the previous example, the message shows a request of product data that has
680 either A001 or A002 as a child part.

```
681 <ProductRecord id="A-7" transaction="07" action="Get" sender="A">  
682 <Condition><Property name="pps:child"><Char value="A001"/></Property></Condition>  
683 <Condition><Property name="pps:child"><Char value="A002"/></Property></Condition>  
684 <Selection type="all"/>  
685 </ProductRecord>
```

686

687 **6.1.4 Selection by wildcard**

688 The third way to select target domain objects is to use wildcard in *Condition* element. To specify the
689 required objects, *wildcard* attribute denotes the property name while the wildcard string is specified in the
690 *value* attribute. The regular expressions are applied for interpreting the wildcard string.

691 Wildcard specification SHOULD only apply to properties that have a value in string format.

692

693 Example 15: Request of customer orders that the destination address has any text of "Boston".

```
694 <CustomerOrder id="A-8" transaction="08" action="Get" sender="A">  
695 <Condition wildcard="pps:delivery" value="Boston"/>  
696 <Selection type="all"/>  
697 </CustomerOrder>
```

698

699 **6.2 Target domain property**

700 When the target domain objects are determined, Get message needs another specification for selecting
701 properties in the domain objects to show the information detail. *Selection* element MAY be used for this
702 purpose. The properties selected by the *Selection* elements are included and the corresponding values
703 are specified by the responder in the Show message.

704 This element MAY represent ordering request/result of the objects in the response message, or
705 calculating request/result of the values of the target objects.

706 **6.2.1 All available properties**

707 When the *type* attribute of *Selection* element has a value of "all", it SHOULD represent that all the
708 possible properties are included in the Show message. The list of properties to return is decided by the
709 responder.

710 When value "typical" is specified in the *type* attribute, the typical properties of the domain object are
711 selected by the responder. The list of typical properties is depending on the domain document. This list is
712 defined by the responder.

713

714 Example 16: Request all the material information. All objects are selected with all possible properties.

```
715 <ResourceRecord id="A-9" transaction="09" action="Get" sender="A">  
716 <Selection type="all"/>  
717 </ResourceRecord>
```

718

719 6.2.2 Selecting domain property

720 In order to specify the properties required in the selected objects, *Property* element in the *Selection*
721 element is used. To select objects, name of property SHOULD be specified in the *name* attribute of
722 *Property* element in the Get message. Property name is defined in the application profile or the
723 implementation profile.

724

725 Example 17: The example shows that the objects in the responding message are required with properties
726 of key, name, and priority.

```
727 <PartyRecord id="A-10" transaction="10" action="Get" sender="A">  
728 <Selection>  
729 <Property name="pps:key"/>  
730 <Propertyv name="pps:name" />  
731 <Property name="pps:priority" />  
732 </Selection>  
733 </PartyRecord>
```

734

735 When the property required has not been defined in the profile, Get message MAY request user-made
736 properties by specifying its own texts following the prefix of "user:". In order to clarify the meaning of the
737 new property, *path* attribute of the property SHOULD be specified in the X-path format that can be used
738 to find data in the primitive elements.

739

740 Example 18: The second example is the case that a user-made property is required. The new property is
741 a particular value resulted by a special calculation applied to a material lot.

```
742 <LotRecord id="A-11" transaction="11" action="Get" sender="A">  
743 <Selection>  
744 <Property name="user:calculation-1"  
745 path="Spec[@type='user:calculation-1']/Qty/@value"/>  
746 </Selection>  
747 </LotRecord>
```

748

749 6.2.3 Sorting by property value

750 Sorting request of the domain objects listed in the Show message MAY be specified in *Property* element
751 in *Selection* element. The *Property* element has *sort* attribute that MAY have a value of "disc" or "asc".
752 The responder who receives this message SHOULD sort the domain objects by descending or ascending
753 order, respectively.

754 When there are more than one *Property* elements in the *Selection* element that has *sort* attribute, first
755 *Property* element is the highest priority of sort. If the values of the property of two objects in the
756 responding domain objects list are the same, then the second data value indicated by the next *Property*
757 element are compared.

758

759 Example 19: Data request with sorting

```
760 <ProductRecord id="A-12" transaction="12" action="Get" sender="A">  
761 <Selection>  
762 <Property name="pps:parent" sort="asc"/>  
763 <Property name="pps:name" sort="asc"/>  
764 </Selection>  
765 </ProductRecord>
```

766

767 Example 20: An example of response of the previous example

```
768 <ProductRecord id="B-12" transaction="12" action="Show" sender="B">  
769 <Item name="bbb"><Compose type="pps:parent" item="A"/></Item>
```

```
770 <Item name="ccc"><Compose type="pps:parent" item="A"/></Item>
771 <Item name="ddd"><Compose type="pps:parent" item="A"/></Item>
772 <Item name="aaa"><Compose type="pps:parent" item="B"/></Item>
773 </ProductRecord>
```

774

775 6.2.4 Calculation of property value

776 *Property* element in a *Selection* element MAY represent a request of calculation of property values that
777 are selected by the Get message. In order to do this, *calc* attribute of *Property* element is used to select a
778 calculation method. The value of *calc* attribute of *Property* element can take either "sum", "ave", "max",
779 "min", and "count" as a calculation method.

780 The name of property that is calculated MAY be specified in *name* attribute of the *Property* element. Then,
781 the values of specified property SHOULD be calculated using the method.

782 In Show message or Notify message, the result of calculation is specified in *Property* element which is in
783 *Header* element. Because Show and Notify element doesn't have *Selection* element, the result need to
784 move from the *Selection* element in the corresponding Get message.

785 The responder who receives this message SHOULD answer by calculating the target property value, and
786 specifies it at the corresponding *value* attribute of the *Property* if the data type is string, or describes it in
787 *Qty* element, *Duration* element, or *Time* element under the *Primitive* element.

788

789 Example 21: Requests to calculate summary of total price

```
790 <CustomerOrder id="A-13" transaction="13" action="Get" sender="A">
791 <Selection>
792 <Property name="pps:price" calc="sum"/>
793 </Selection>
794 <Selection type="all"/>
795 </CustomerOrder>
```

796

797 Example 22: The corresponding response of the previous example

```
798 <CustomerOrder id="B-13" transaction="13" action="Show" sender="B">
799 <Header count="3">
800 <Property name="pps:price" calc="sum"><Price value="2500"/></Property>
801 </Header>
802 <Order id="001" item="Product-1"><Price><Qty value="1000" unit="USD"/></Price></Order>
803 <Order id="004" item="Product-1"><Price><Qty value="1000" unit="USD"/></Price></Order>
804 <Order id="007" item="Product-1"><Price><Qty value="500" unit="USD"/></Price></Order>
805 </CustomerOrder>
```

806

807 The response message to the calculation request has the calculation result in *Property* element in *Header*
808 element. If the calculation method is "count", then the result value is the number of corresponding domain
809 objects in the database. In order to know the number of data before the detailed query execution, this
810 calculation request MAY be send without *Selection* element that shows the property items in the Show
811 message. In the case that "count" value is specified in type attribute, name attribute of *Property* element
812 MAY NOT be specified.

813

814 Example 23-A: Request of counting the number of data

```
815 <CustomerOrder id="A-14" transaction="14" action="Get" sender="A">
816 <Selection>
817 <Property calc="count"/>
818 </Selection>
819 </CustomerOrder>
```

820

821 Example 23-B: The answer of the request of counting the data

```
822 <CustomerOrder id="B-14" transaction="14" action="Show" sender="B">  
823 <Header>  
824 <Property calc="count"><Qty value="55"/></Property>  
825 </Header>  
826 </CustomerOrder>
```

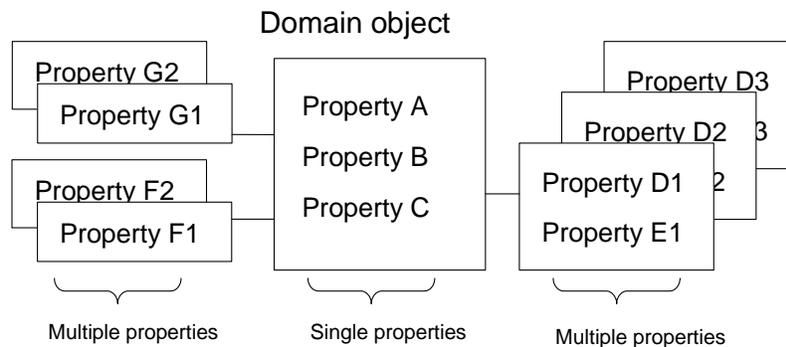
827

828 6.3 Multiple property

829 A transaction element for a simple Get message has one *Selection* element that has several properties
830 required. However, if the target domain object has a multiple property and some of its instances need to
831 be selected, each multiple property SHOULD have corresponding *Selection* element. The *Selection*
832 element for the multiple property needs *Condition* element as its child element to represent conditions to
833 select the instances.

834 From a modeling perspective, a multiple property can be defined in the attribute objects that are
835 associated with or contained by the domain object. The target domain object and attribute objects has
836 one-to-many relations. The figure shows that Property A, B, and C is single type, while Property D to G
837 are multiple. In this figure, it is important that Property D and E are on the same attribute object, and any
838 conditions for property selection are applied in the same manner.

839



840

841 *Figure 14 Single property and Multiple property*

842

843 In accordance with this conceptual structure, a *Selection* element SHOULD be defined for each attribute
844 class, i.e. type of attribute objects. For example, the case of the figure can have three different *Selection*
845 elements, one of which has Property D and Property E at the same time.

846

847 Example 24-A: A request of calendar information of a customer in April.

```
848 <PartyRecord id="A-15" transaction="15" action="Get" sender="A">  
849 <Condition id="001"/>  
850 <Selection>  
851 <Property name="pps:id" />  
852 <Property name="pps:name"/>  
853 </Selection>  
854 <Selection>  
855 <Property name="pps:calendar-date" />  
856 <Property name="pps:calendar-value"/>  
857 <Condition>  
858 <Property name="pps:calendar-date">  
859 <Time value="2006-04-01T00:00:00" condition="earliest"/>  
860 </Property>  
861 <Property name="pps:calendar-date">  
862 <Time value="2006-05-01T00:00:00" condition="latest" exclusive="true"/>  
863 </Property>  
864 </Condition>
```

```
865 </Selection>
866 </PartyRecord>
```

867

868 Example 24-B: One possible answer to the previous message.

```
869 <PartyRecord id="B-15" transaction="15" action="Show" sender="B">
870 <Party id="001">
871 <Capacity status="pps:holiday"><Time value="2006-04-01T00:00:00"/></Capacity>
872 <Capacity status="pps:work"><Time value="2006-04-02T00:00:00"/></Capacity>
873 <Capacity status="pps:work"><Time value="2006-04-03T00:00:00"/></Capacity>
874 ...
875 <Capacity status="pps:work"><Time value="2006-04-30T00:00:00"/></Capacity>
876 </Party>
877 </PartyRecord>
```

878

879 When there is more than one *Selection* element in a transaction element, the first *Selection* element
880 SHOULD NOT have *Condition* element. The *Selection* element that selects multiple properties SHOULD
881 be specified at the second or later.

882 6.4 Using Header element

883 6.4.1 Query by header element

884 In a *Header* element of a Get message, brief query information can be added independent from the main
885 query mechanism provided by *Condition* and *Selection* elements. The brief query mechanism is activated
886 when *id* attribute of *Header* element in a Get message has an ID.

887 The responder to this message SHOULD get the corresponding domain object that has the ID, and
888 answer its property values required by Primitive elements of *Header* element in the Get message. The
889 Primitive elements for the brief query have *type* attribute that has "target" value, or the attribute doesn't
890 have a value because "target" is default value.

891 The target object in this brief query is basically in the same class of the domain objects, unless the
892 *transaction* attribute of *Header* element has another name of transaction element. When the *transaction*
893 attribute is specified a name of another domain object, the corresponding information of the domain
894 objects will be answered in the *Header* of the Show message.

895 Multiple property MAY not be performed properly in this mechanism because the answer is formatted in
896 single type properties. If a multiple property is selected in the *Header*, arbitrarily instance of the property
897 is specified in the answer message.

898

899 Example 25: *Header* element for brief query has *id* attribute that is specified a name of the object.

```
900 <ProductRecord id="A-16" transaction="16" action="Get" sender="A">
901 <Header id="001">
902 <Property type="target" name="pps:name"/>
903 </Header>
904 </ProductRecord>
```

905

906 Example 26: An answer of the previous message

```
907 <ProductRecord id="B-16" transaction="16" action="Show" sender="B">
908 <Header id="001">
909 <Property type="target" name="pps:name"><Char value="Product-A"/></Property>
910 </Header>
911 </ProductRecord>
```

912

913 6.4.2 Count of domain objects

914 In Get message, *count* attribute of *Selection* element SHOULD represent the maximum number of objects
915 requested in the response message. If the value of the *count* attribute is 1 or more than 1, then the
916 number specified in the attribute restricts the size of the response message.

917 When many domain objects are in the database, they can be retrieved separately by several Get
918 messages. In such case, *offset* attribute of *Selection* element SHOULD be specified as an offset number
919 to skip the first objects while reading the domain objects.

920 The offset request MAY be effective when a sort mechanism performed according to the value of *sort*
921 attribute in *Property* element. If there is no description of sort, then the application MAY concern that the
922 domain objects are sorted by the values of their IDs.

923 The attribute of *count* and *offset* SHOULD NOT be specified if the *Selection* element is the second or
924 later description in the transaction element. In the corresponding Show message, the attribute of *count*
925 and *offset* are specified in the *Header* element instead of *Selection* element.

926

927 Example 27: The following message requests customer order from #101 to #110.

```
928 <CustomerOrder id="A-17" transaction="17" action="Get" sender="A">  
929 <Selection offset="100" count="10"/>  
930 <Property name="pps:id" sort="desc"/>  
931 </Selection>  
932 </CustomerOrder>
```

933

934 6.5 Show message

935 6.5.1 Structure of Show message

936 Show message has the same structure as the structure of Notify message. This message SHOULD have
937 a value of "Show" at the *action* attribute.

938 Show message SHOULD have header information by *Header* element, and if the Get message requests
939 calculation by specifying *calc* attribute of *Selection* elements, then the calculation results SHOULD be
940 specified in *Header* element.

941 Body of Show messages SHOULD have all the content of the domain objects that corresponds to the
942 request. The body MAY be empty if the corresponding object doesn't exist.

943

944 Example 28: The message of customer order #001 that has total amount and detailed item lists.

```
945 <CustomerOrder id="B-18" transaction="18" action="Show" sender="B">  
946 <Header id="001" count="3" title="OrderSheet">  
947 <Property name="pps:party" value="K-Inc." display="CSTM"/>  
948 <Property type="selection" name="pps:id" display="PN"/>  
949 <Property type="selection" name="pps:name" display="NAME"/>  
950 <Property type="selection" name="pps:qty" display="QTY"/>  
951 <Property type="selection" calc="sum" name="pps:price" display="PRICE">  
952 <Qty value="1200"/></Property>  
953 </Header>  
954 <Order id="001-1" item="Product-A1"><Qty value="1"/></Order>  
955 <Order id="001-2" item="Product-A2"><Qty value="10"/></Order>  
956 <Order id="001-3" item="Product-A3"><Qty value="3"/></Order>  
957 </CustomerOrder>
```

958

959 6.5.2 Header in Show message

960 In Show messages, the number of domain objects listed in the body of the message is specified as the
961 value of *count* attribute of the *Header* element.

962 *Property* elements specified in the *Header* element consist of three types. First type is for properties of a
963 header domain object requested by the Get message as header query. All *Property* elements of this
964 group SHOULD have a value "target" at the *type* attribute or the attribute is not specified. This property
965 represents any value of the header object selected by *id* attribute of the *Header* element.

966 The second type of *Property* elements has a value "condition" at the *type* attribute. This property
967 SHOULD represent that all domain objects listed in the body of the message has the same value
968 specified in the property. Application program who responses the Show message MAY create the
969 properties simply by duplicating the corresponding *Property* elements in *Condition* element of the Get
970 message, because the property is the condition of the domain objects.

971 The final group of properties comes from the *Selection* element of the Get message. The properties in
972 this group SHOULD have a value "selection" at the *type* attribute. These properties are basically a copy of
973 *Property* elements of the *Selection* element in the Get message. If the *Selection* element in the Get
974 message requests calculation, the results are added in the *value* attribute or *Qty*, *Duration*, *Time* sub-
975 element of the *Property* element. In addition, a value of *display* attribute MAY be specified for any texts in
976 the header area for printing a formatted sheet.

977

978 Example 29: A request to get product information of "A001" and its parts list.

```
979 <ProductRecord id="A-19" transaction="19" action="Get" sender="A">  
980 <Condition>  
981 <Property name="pps:parent" vaue="A001"/>  
982 </Condition>  
983 <Selection>  
984 <Property name="pps:id"/>  
985 <Property name="pps:name"/>  
986 </Selection>  
987 <Header title="BillOfMaterials" id="A001" >  
988 <Property name="pps:name"/>  
989 <Property name="pps:price"/>  
990 <Property name="pps:price-unit"/>  
991 </Header>  
992 </ProductRecord>
```

993

994 Example 30: The response to the previous Get message. The *Property* elements in the *Condition* element
995 and *Selection* element are copied and specified as a child element of the header, while the values of the
996 header object information is added.

```
997 <ProductRecord id="B-19" transaction="19" action="Show" sender="B">  
998 <Header title="BillOfMaterials" id="A001" count="3">  
999 <Property name="pps:name"><Char value="Product A001"/></Property>  
1000 <Property name="pps:price"><Qty value="2000"/></Property>  
1001 <Property name="pps:price-unit"><Char vaue="yen"/></Property>  
1002 <Property type="condition" name="pps:parent"><Char vaue="A001"/></Property>  
1003 <Property type="selection" name="pps:id"/>  
1004 <Property type="selection" name="pps:name"/>  
1005 </Header>  
1006 <Item id="A001-01" name="Part A001-01"/>  
1007 <Item id="A001-02" name="Part A001-02"/>  
1008 <Item id="A001-03" name="Part A001-03"/>  
1009 </ProductRecord>
```

1010

1011

1012 7 XML Elements

1013 7.1 Error element

1014 Error information SHOULD be specified in the error element under the transaction element when one
1015 application program needs to send the error results to the requester. The error elements MAY be
1016 specified in Show messages and Confirm messages.

1017 The transaction element SHOULD have one or more Error elements if the message is sent as error
1018 information. The transaction element SHOULD NOT have an Error element if the message is a normal
1019 response in the messaging models.

1020 This information SHOULD be specified in the following XML schema. The XML documents generated by
1021 the schema SHOULD be consistent with the following arguments.

1022

```
1023 <xsd:element name="Error">  
1024 <xsd:complexType>  
1025 <xsd:attribute name="id" type="xsd:string"/>  
1026 <xsd:attribute name="ref" type="xsd:string"/>  
1027 <xsd:attribute name="code" type="xsd:string"/>  
1028 <xsd:attribute name="location" type="xsd:string"/>  
1029 <xsd:attribute name="status" type="xsd:string"/>  
1030 <xsd:attribute name="description" type="xsd:string"/>  
1031 </xsd:complexType>  
1032 </xsd:element>
```

1033

- 1034 ● *id* attribute SHOULD represent identifier that application can identify the error data.
- 1035 ● *ref* attribute SHOULD represent the transaction message id that has the errors.
- 1036 ● *code* attribute SHOULD represent unique identifier of the error categories. The error code SHOULD
1037 consist of three digits. If the first digit is 0, then the code SHOULD represent as follows:
 - 1038 ➤ "000" represents "Unknown error".
 - 1039 ➤ "001" represents "Connection error".
 - 1040 ➤ "002" represents "Authorization error".
 - 1041 ➤ "003" represents "Application is not ready".
 - 1042 ➤ "004" represents "Message buffer is full".
 - 1043 ➤ "005" represents "Syntax error (communication)".
 - 1044 ➤ "006" represents "Syntax error (application logic)".
 - 1045 ➤ "007" represents "Requested task is not supported".
 - 1046 ➤ "008" represents "Requested task is denied".
 - 1047 ➤ "009" represents "No data object requested in the message".
 - 1048 ➤ "010" represents "Data object requested already exists".
 - 1049 ➤ "011" represents "Application error".
 - 1050 ➤ "012" represents "Abnormal exception".
- 1051 ● *location* attribute SHOULD represent the location of error texts.
- 1052 ● *status* attribute SHOULD represent a status. Values of this attribute SHOULD include:
 - 1053 ➤ "error" represents that the message is error notification.
 - 1054 ➤ "warning" represents that the message is warning.
- 1055 ● *description* attribute SHOULD represent any description of the error explanations.

1056 7.2 App element

1057 Application information MAY be used by application programs by their own ways. For this purpose, App
1058 element is defined. App element is extension area for application programs who may want to have their
1059 own information by using another name spaces. If the application programs within a messaging model
1060 can decide to have a new namespace, they have their own XML schema for the App element.

1061 This element SHOULD be consistent with the following XML schema.

1062

```
1063 <xsd:element name="App">  
1064 <xsd:complexType>  
1065 <xsd:sequence>  
1066 <xsd:any minOccurs="0" maxOccurs="unbounded"/>  
1067 </xsd:sequence>  
1068 </xsd:complexType>  
1069 </xsd:element>
```

1070

1071 7.3 Condition element

1072 *Condition* element SHOULD represent any condition to select domain objects or domain properties. The
1073 conditions can be defined by *Property* elements, which can represent value or range of values.

1074 If there are more than one *Condition* element in the same XML element, then these conditions SHOULD
1075 be regarded disjunctive manner.

1076 This information SHOULD be specified in the following XML schema. The XML documents generated by
1077 the schema SHOULD be consistent with the following arguments.

1078

```
1079 <xsd:element name="Condition">  
1080 <xsd:complexType>  
1081 <xsd:sequence>  
1082 <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>  
1083 </xsd:sequence>  
1084 <xsd:attribute name="id" type="xsd:string"/>  
1085 <xsd:attribute name="wildcard" type="xsd:string"/>  
1086 <xsd:attribute name="value" type="xsd:string"/>  
1087 <xsd:attribute name="version" type="xsd:string"/>  
1088 </xsd:complexType>  
1089 </xsd:element>
```

1090

1091 ● *Property* element SHOULD represent any properties that restrict the target objects by describing a
1092 value or range of value.

1093

1094 ● *id* attribute SHOULD represent the identifier of the target domain object. When the target object is
1095 known, then this value is specified instead of describing any other conditions.

1096 ● *wildcard* attribute SHOULD represent the name of property that is used to apply wildcard value.
1097 The wildcard text is specified in the *value* attribute.

1098 ● *value* attribute SHOULD represent the wildcard text for selecting the target domain objects. The
1099 text is interpreted by regular expression rules.

1100 ● *version* attribute SHOULD represent version name of the target object. The format of version texts
1101 is managed in application programs. Values of this attribute MAY include:

1102 ➤ "latest" --- the latest version object

1103 ➤ "earliest" – the earliest version object

1104 ➤ any string that represent a version identifier

1105

1106 7.4 Selection element

1107 *Selection* element SHOULD represent information which property is selected in domain properties in the
1108 domain object. *Selection* elements are used in Get transactions and Change transactions.

1109 In Change transactions, *Selection* element is used to select the property that the requester tries to
1110 change the value. In Get transactions, *Selection* element is used to select the target properties to select
1111 in the Show message. If there is no *Select* element in Get message, then the corresponding Show
1112 message doesn't have any domain objects in its message body.

1113 This information SHOULD be specified in the following XML schema. The XML documents generated by
1114 the schema SHOULD be consistent with the following arguments.

1115

```
1116 <xsd:element name="Selection">  
1117 <xsd:complexType>  
1118 <xsd:sequence>  
1119 <xsd:element ref="Condition" minOccurs="0" maxOccurs="unbounded"/>  
1120 <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>  
1121 </xsd:sequence>  
1122 <xsd:attribute name="type" type="xsd:string"/>  
1123 <xsd:attribute name="multi" type="xsd:int"/>  
1124 <xsd:attribute name="count" type="xsd:int"/>  
1125 <xsd:attribute name="offset" type="xsd:int"/>  
1126 </xsd:complexType>  
1127 </xsd:element>
```

1128

1129 ● *Condition* element SHOULD represent any condition for selecting members of a multiple property,
1130 when the *multiple* attribute is greater than 1. Change or Get message can restrict its target by this
1131 condition.

1132 ● *Property* element SHOULD represent any property required to describe in the target domain
1133 objects. In the case of Get-Show model, the corresponding information of this property is
1134 addressed in the body of the response message.

1135

1136 ● *type* attribute SHOULD represent the type of action after selecting the target properties. The
1137 available values are defined depending on the type of message.

1138 ➤ “insert” for Change message represents that the property value is inserted, this is default value,

1139 ➤ “update” for Change message represents that the property value is updated,

1140 ➤ “delete” for Change message represents that the property value is deleted,

1141 ➤ “none” for Get message represents that the target is specified by *Property* element, this is
1142 default value,

1143 ➤ “typical” for Get message represents that the target property is typical set,

1144 ➤ “all” for Get message represents that the target property is all properties in the object.

1145 ● *multi* attribute SHOULD show whether the selected property is multiple or single one. When the
1146 property is single, the value of “type” attribute in Change message is not necessary because the
1147 operation is always updating.

1148 ● *count* attribute

1149 ● *offset* attribute

1150

1151 7.5 Header element

1152 *Header* element is used for representing header information in Show and Notify messages. The header
1153 information is specified for any data depending on the document as a whole. In Get message, *Header*

1154 element MAY be used to make brief query of domain object that is not in the target of domain document.
1155 The *Header* element SHOULD be specified in the transaction element.
1156 This information SHOULD be specified in the following XML schema. The XML documents generated by
1157 the schema SHOULD be consistent with the following arguments.

1158

```
1159 <xsd:element name="Header">  
1160 <xsd:complexType>  
1161 <xsd:sequence>  
1162 <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>  
1163 </xsd:sequence>  
1164 <xsd:attribute name="id" type="xsd:string"/>  
1165 <xsd:attribute name="class" type="xsd:string"/>  
1166 <xsd:attribute name="title" type="xsd:string"/>  
1167 <xsd:attribute name="count" type="xsd:int"/>  
1168 <xsd:attribute name="offset" type="xsd:int"/>  
1169 </xsd:complexType>  
1170 </xsd:element>
```

1171

1172 ● *Property* element SHOULD represent any property of the target object in the header or any
1173 aggregation value of domain objects in the body of the message.

1174

1175 ● *id* attribute SHOULD represent ID of the target object that is shown in the header by representing
1176 its property in the “Property” element.

1177 ● *class* attribute SHOULD represent the target domain object that the header shows the information
1178 in *Property* elements. Default value is the name of domain object the document refers as default.

1179 ● *title* attribute SHOULD represent a title of the message document.

1180 ● *count* attribute SHOULD represent the number of domain objects in the message. When this
1181 attribute is used in Confirm message and Show message, the value equals to the number of object
1182 in the body of message. In Get message, the value represents the maximum number the receiver
1183 is expecting the objects in the Show message.

1184 ● *offset* attribute SHOULD represent the offset number of data list. When the objects in the message
1185 is not all of the existing objects in the sender, the offset value shows the relative position of the first
1186 object on the message body in the whole objects. This attribute can be used in Get message as a
1187 request to offset the response data.

1188

1189 7.6 Property element

1190 *Property* element represents property information of domain objects under *Condition* element, *Selection*
1191 element and *Header* element. When *Condition* element has a *Property* element, it shows condition of
1192 selecting the domain object. When *Selection* element has a *Property* element, it shows the target property
1193 of changing or getting messages. When *Header* element has a *Property* element, it shows a property of
1194 the header object or aggregation information of the body objects.

1195 This information SHOULD be specified in the following XML schema. The XML documents generated by
1196 the schema SHOULD be consistent with the following arguments.

1197

```
1198 <xsd:element name="Property">  
1199 <xsd:complexType>  
1200 <xsd:choice>  
1201 <xsd:element ref="Qty" minOccurs="0" maxOccurs="unbounded"/>  
1202 <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>  
1203 <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>  
1204 </xsd:choice>  
1205 <xsd:attribute name="type" type="xsd:string"/>  
1206 <xsd:attribute name="name" type="xsd:string"/>  
1207 <xsd:attribute name="path" type="xsd:string"/>
```

1208
1209
1210
1211
1212
1213

1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255

```
<xsd:attribute name="value" type="xsd:string"/>  
<xsd:attribute name="sort" type="xsd:string"/>  
<xsd:attribute name="calc" type="xsd:string"/>  
<xsd:attribute name="display" type="xsd:string"/>  
</xsd:complexType>  
</xsd:element>
```

- *Qty*, *Char*, and *Time* elements SHOULD represent the value of the primitive. If the data type of the value is unknown, the *value* attribute can be used instead of the elements.

- *type* attribute SHOULD represent a type of property. This attribute is used only when the *Property* element is defined under the *Header* element. The value of this attribute is one of the followings:
 - “target” --- the property of the header target object,
 - “condition” --- the condition data of the objects in the body. This data is copied from the property data in the *Condition* element.
 - “selection” --- the selection data of the properties of objects in the body. This data is copied from the property data in the *Selection* element.

- *name* attribute SHOULD represent a name of property. The value of this attribute is the string that is defined in the corresponding profile or a name of user-extended property whose name is starting with “user:”.

- *path* attribute SHOULD represent X-path string that shows the position of the data in the corresponding primitive element. This attribute is required only if the value of the “name” attribute shows that the property is user-extended property, because such path data is predefined in the profile for the others.

- *value* attribute SHOULD represent the value of property. When the data type of this attribute is unknown, the data represents the value instead of *Qty*, *Char* and *Time* elements.

- *sort* attribute SHOULD represent that the objects in the body of this message are expected to be sorted by ascending or descending order. For *Get* message, this attribute SHOULD be used in the *Property* element under *Selection* element. For *Show* message and *Notify* message, the *Property* element SHOULD be specified in *Header* element. If more than one *Property* element that has *sort* attribute are specified, these *sort* requests SHOULD be applied in the priority rule that faster element dominate the followers. This attribute SHOULD NOT use together with the *calc* attribute.
 - “asc” --- sort in ascending order,
 - “desc” --- sort in descending order.

- *calc* attribute SHOULD represent that the property is expected to be calculated for the objects in the body of this message. For *Get* message, this attribute SHOULD be used in the *Property* element under *Selection* element. For *Show* message and *Notify* message, the *Property* element SHOULD be specified in *Header* element. This attribute SHOULD NOT use together with the *sort* attribute.
 - “sum” --- summary of the value of properties of the target objects,
 - “ave” --- average of the value of properties of the target objects,
 - “max” --- maximum value of properties of the target objects,
 - “min” --- minimum value of properties of the target objects,
 - “count” --- the number of the target objects in the body.

- *display* attribute SHOULD represent the string that can be shown in the header line for each primitive as an explanation. This attribute is used only under the *Header* element in *Show* messages and *Notify* messages.

1256

A. Implementation level

1257 Since this part of standard provides application programs a height level functionality in information
1258 exchange on planning and scheduling problems, it might be hard to implement for the application
1259 programs that don't need full capability of messaging. For this situation, this part defines implementation
1260 levels for each functional category.

1261 The implementation level is specified in implementation profiles defined in [PPS03]. Each application
1262 program MAY describe its capability for each messaging model. Therefore, system designer of the
1263 domain problem knows available combination of messaging without making a configuration tests.

1264 The following table prescribes the implementation levels.

1265

1266 *Table 5 Implementation levels*

Level	Description
0	The application program has no capability of the function
1	The application program has some capability of the function. The partial function is defined for the restricted specifications.
2	The application program has all capability on the function prescribed in this standard

1267

1268 There are some functional categories of specifications, in which some additional constraints MAY be add
1269 to restrict the full specification. The level 1 of implementation is conformed to this restricted specification.
1270 The Table 6 shows the functionality and the level 1 prescription for each category.

1271

1272 *Table 6 Additional constraints to restrict capability of the specification*

Functionality	Description
Multiple documents (see 3.3)	Full Functions: One message MAY have more than one domain documents. Level 1 Implementation: One message SHOULD have one domain document. The second document and others are ignored.
Multiple property identification (see 4.2.2, 4.2.3, 6.3)	Full Functions: Get message MAY have more than one <i>Selection</i> elements. A <i>Selection</i> element MAY have <i>Condition</i> elements. Change message MAY have <i>Selection</i> element that has <i>Condition</i> elements. Level 1 Implementation: Get message SHOULD NOT have more than one <i>Selection</i> elements. A <i>Selection</i> element SHOULD NOT have <i>Condition</i> elements. Query of partial numbers of multiple properties is not available. Update or delete of particular property in multiple property is not available.
Calculation for Show messages (see 6.2.3 and 6.2.4)	Full Functions: <i>Property</i> element MAY have <i>sort</i> attribute and <i>calc</i> attribute to request sorting and calculation of domain objects in the Show message. Level 1 Implementation: <i>Property</i> that has sort or calc attribute SHOULD NOT be specified in Get messages. Sort and Calculation requests in Get transaction are not available.
Query by <i>Header</i> element (see 6.4.1)	Full Functions: Get message MAY have <i>Header</i> element to query particular header object and/or to restrict the total number of domain objects in the Show message.

	Level 1 Implementation: Get message SHOULD NOT have <i>Header</i> element. Query by header object is not available. Restriction of domain object in Show message is not available.
--	--

1273

1274

1275 **B. Acknowledgements**

1276 The following individuals have participated in the creation of this specification and are gratefully
1277 acknowledged:

1278 **Participants:**

- 1279 Shinya Matsukawa, Hitachi
- 1280 Tomohiko Maeda, Fujitsu
- 1281 Masahiro Mizutani, Unisys Corporation
- 1282 Akihiro Kawauchi, Individual Member
- 1283 Yuto Banba, PSLX Forum
- 1284 Osamu Sugi, PSLX Forum
- 1285 Hideichi Okamune, PSLX Forum
- 1286 Hiroshi Kojima, PSLX Forum
- 1287 Ken Nakayama, Hitachi
- 1288 Yukio Hamaguchi, Hitachi
- 1289 Tomoichi Sato, Individual
- 1290 Hiroaki Sasaki, Individual

1291

1292

C. Revision History

1293

Revision	Date	Editor	Changes Made

1294