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

Public Review Draft 01

7 August 2007

Specification URIs:

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

Previous Version:

N/A

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

Latest Approved Version:

N/A

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/transaction-messages

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

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", [insert specific trademarked names and abbreviations here] 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

Figures

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/

1.4 Conformance

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

http://docs.oasis-open.org/pps/v1.0/pps-transaction-messages.xsd

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

1.5 Terms and definitions

Messaging model

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

Primitive element

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

Transaction element

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

Domain document

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

Domain object

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

Domain property

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

Implementation profile

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

Application profile definition

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

2 Messaging model

2.1 Basic models

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

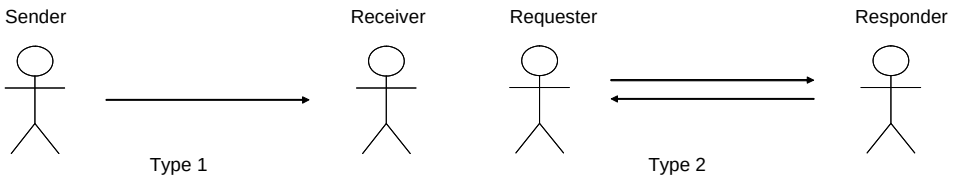


Figure 1 Basic types of messaging

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

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

2.2 Message classes

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

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.

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

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.

2.3 Messaging models

2.3.1 Add transaction

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

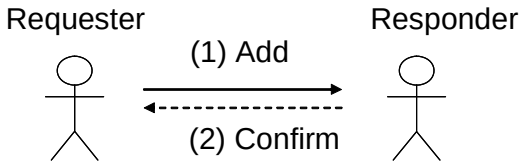


Figure 2 Add transaction

2.3.2 Change transaction

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

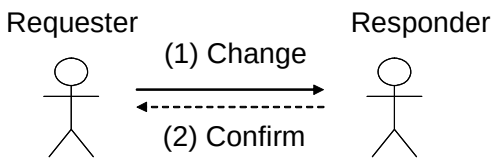


Figure 3 Change transaction

2.3.3 Remove transaction

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

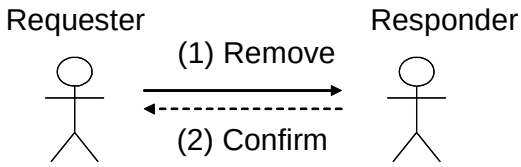


Figure 4 Remove transaction

2.3.4 Notify transaction

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

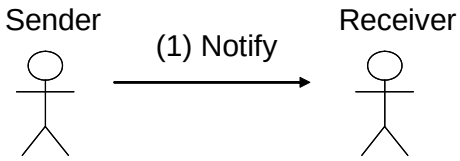


Figure 5 Notify transaction

2.3.5 Sync transaction

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

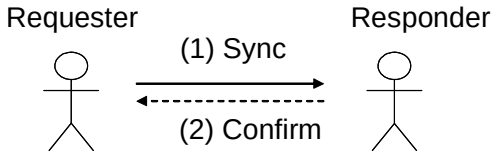


Figure 6 Sync transaction

2.3.6 Get-Show transaction

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

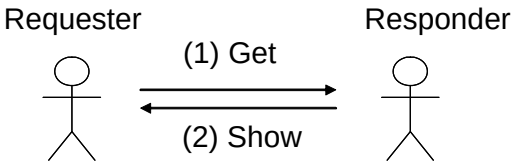


Figure 7 Get-Show transaction

2.4 Procedures on responders

2.4.1 Common tasks

Responders SHOULD have the capability to perform the following tasks when a massage document is received.

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

2.4.2 Confirm message

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

- The responder SHOULD send a Confirm message to the requester when the Add message received has an attribute of confirm="Always". The Confirm message SHOULD have either error information or the id list that shows all the objects added to the database.
- The responder SHOULD send a Confirm message to the requester when the Change message received has an attribute of confirm="Always". The Confirm message SHOULD have either error information or the id list that shows all the objects changed in the database.
- The responder SHOULD send a Confirm message to the requester when the Remove message received has an attribute of confirm="Always". The Confirm message SHOULD have either error information or the id list that shows all the objects deleted in the database.
- The responder SHOULD send a Confirm message to the requester when the Sync message received has an attribute of confirm="Always". The Confirm message SHOULD have either error information or the id list that shows all the objects to be monitored for synchronization.

2.4.3 Error handling

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

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

- The responder who receives an Add message with the attribute of confirm=”OnError” and is hard to respond in normal processes because of errors, SHOULD send a Confirm message with the error information to the requester.
- The responder who receives a Change message with the attribute of confirm=”OnError” and is hard to respond in normal processes because of errors, SHOULD send a Confirm message with the error information to the requester.
- The responder who receives a Remove message with the attribute of confirm=”OnError” and is hard to respond in normal processes because of errors, SHOULD send a Confirm message with the error information to the requester.
- The responder who receives a Sync message with the attribute of confirm=”OnError” and is hard to respond in normal processes because of errors, SHOULD send a Confirm message with the error information to the requester.

3 Message document

3.1 Message Structure

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

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

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

3.2 Transaction element

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

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

```
<xsd:complexType name="TransactionType">
  <xsd:sequence>
    <xsd:element ref="Error" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="App" minOccurs="0"/>
    <xsd:element ref="Spec" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Condition" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Selection" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Header" minOccurs="0"/>
    <xsd:choice minOccurs="0">
      <xsd:element ref="Party" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Plan" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Order" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Item" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Resource" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Process" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Lot" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Task" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Operation" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="ref" type="xsd:string"/>
  <xsd:attribute name="action" type="xsd:string"/>
  <xsd:attribute name="transaction" type="xsd:string"/>
  <xsd:attribute name="confirm" type="xsd:string"/>
  <xsd:attribute name="profile" type="xsd:string"/>
  <xsd:attribute name="sender" type="xsd:string"/>
  <xsd:attribute name="create" type="xsd:dateTime"/>
  <xsd:attribute name="description" type="xsd:string"/>
</xsd:complexType>
```

- *id* attribute SHOULD represent the identifier of the message. Every transaction message SHOULD have a unique id in the scope of the sender or the requester.
- *ref* attribute SHOULD represent the identifier of a primary message document or other message document that is in the same message, when the message has more than one message document.

- *action* attribute SHOULD represent the type of the message, where the types correspond to verbs information for the message. Values of the attribute SHOULD be either “Add”, “Change”, “Remove”, “Confirm”, “Notify”, “Sync”, “Get”, or “Show”.
- *transaction* attribute SHOULD represent the identifier of the series of a transaction. If the message has a predecessor that relates to the message, the values of this attribute in those messages SHOULD be the same. For example, messages in the same massaging model SHOULD have the same value on the transaction attribute. Re-sending messages SHOULD have the same value of transaction attribute with the original message.
- *confirm* attribute SHOULD represent a confirmation request. The value of the attribute MSUT be either “Never”, “OnError”, or “Always”.
- *profile* attribute SHOULD identify the application profile data, which describes details of domain objects and domain properties to adjust the particular business applications. The empty value of the attribute SHOULD represent the transaction follows the standard profile.
- *sender* attribute SHOULD represent an identifier of the sender or requester of the message. This information is not for the low-level communication programs but for application programs.
- *create* attribute SHOULD represent a date when the transaction document is created.
- *description* attribute SHOULD represent any comments or descriptions.

Elements under the transaction element SHOULD follow the sentences:

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

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

Table 3 Structure of transaction element

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

3.3 Multiple documents message

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

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

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

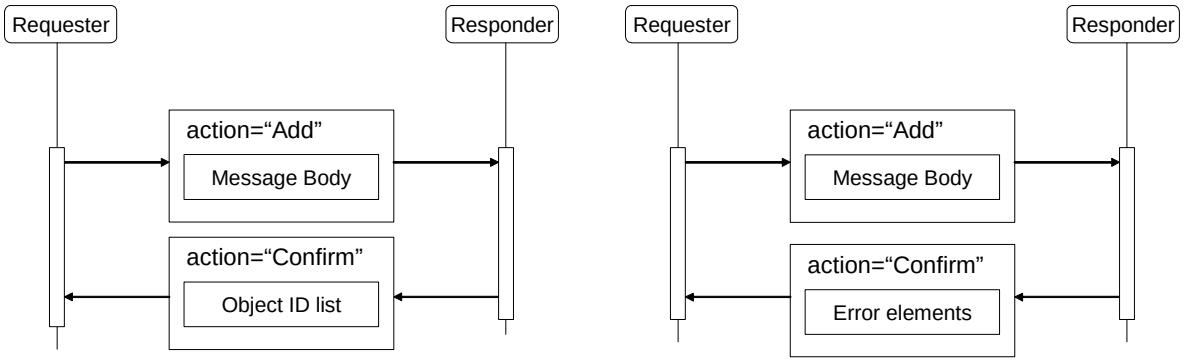
4 Add, Change and Remove transaction

4.1 Add transaction

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

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

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



(a) Normal case (b) Error case

Figure 8 Add message transactions

Example 1-A: Message to add three Product Records

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

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

Example1-B: Add message using a *Condition* element

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

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

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

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

4.2 Change transaction

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

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

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

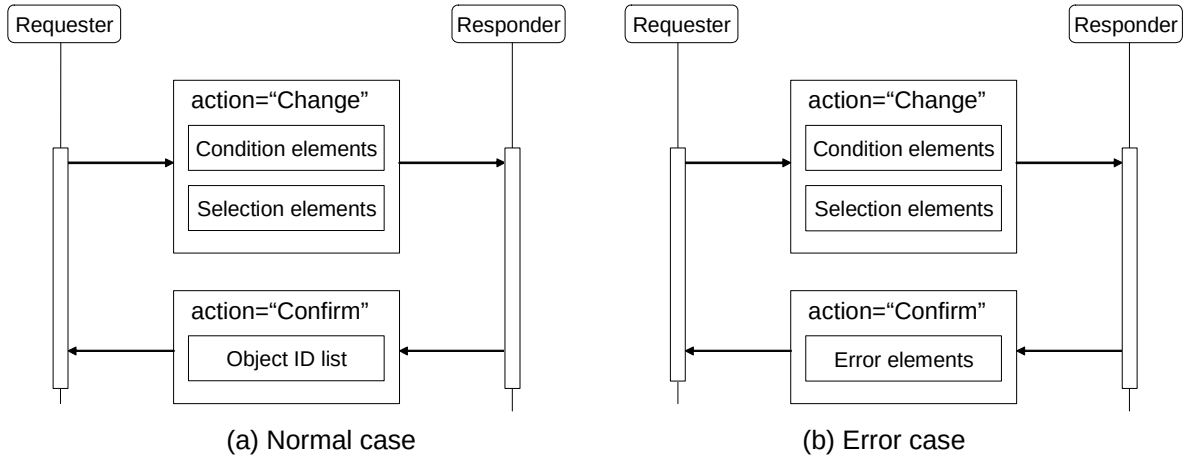


Figure 9 Change message transactions

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

4.2.1 Insert property

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

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

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

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

4.2.2 Update property

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

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

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

```
<ProductRecord id="A-5" transaction="05" action="Change" sender="A">
<Condition id="A001"/>
<Selection type="update" >
  <Property name="pps:child-value"><Qty value="4"/></Property>
  <Condition name="pps:child" value="A001-2"/>
</Selection>
</ProductRecord>
```

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

```
<Item id="A001">
<Compose type="pps:child" value="1" item="A001-1"/>
<Compose type="pps:child" value="1" item="A001-2"/>
<Compose type="pps:child" value="1" item="A001-3"/>
</Item>
```

Example 3-C: Revised status of the product data

```
<Item id="A001">
<Compose type="pps:child" value="1" item="A001-1"/>
<Compose type="pps:child" value="4" item="A001-2"/>
<Compose type="pps:child" value="1" item="A001-3"/>
</Item>
```

4.2.3 Delete property

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

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

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

```
<ProcessRecord id="A-6" transaction="06" action="Change" sender="A">
<Conditoin id="Proc-01"/>
<Select type="delete">
   <Condition name="pps:equipment" value="Machine-1"/>
</Selection>
```

```
</ProcessRecord>
```

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

```
<InventoryRecord id=”A-7” transaction=”07” action=”Change” sender=”A”>
<Condition id=”A001”/>
<Selection type=”delete”>
  <Condition name=”pps:stock-date”>
    <Time value=”2006-08-01T00:00:00” condition=”latest”/>
  </Condition>
</Selection>
</InventoryRecord>
```

4.3 Remove transaction

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.

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

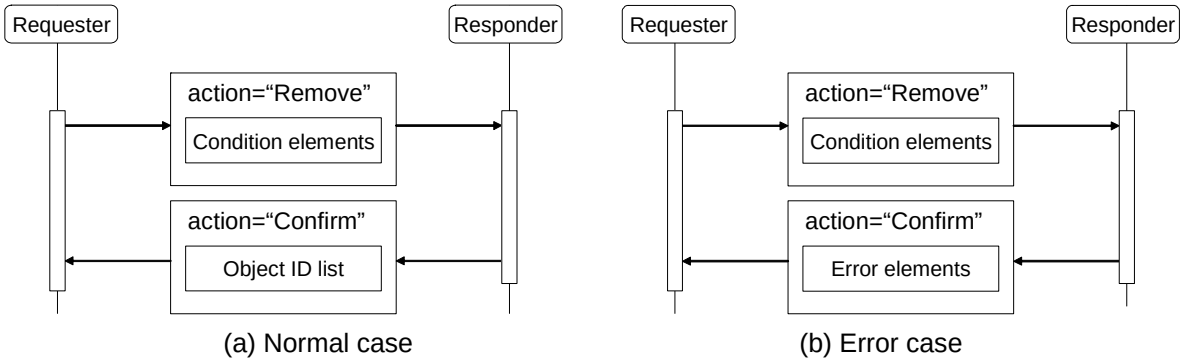


Figure 10 Remove message transactions

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

```
<LotSchedule id=”A-8” transaction=”08” action=”Remove” sender=”A”>
<Condition>
  <Property name=”pps:item” value=”M001”/>
</Condition>
</LotSchedule>
```

5 Notify and Sync Transactions

5.1 Notify transaction

Notify message SHOULD have “Notify” in the *action* attribute. The figure shows that transaction pattern of Notify message exchange. Notify message doesn’t need its response.

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

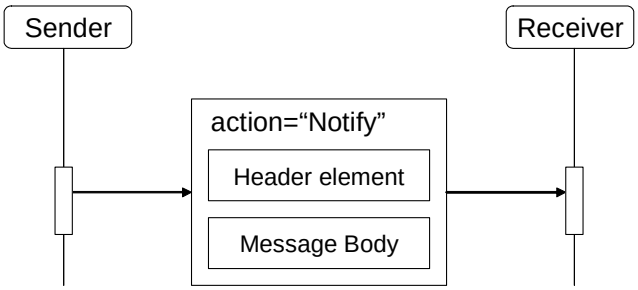


Figure 11 Notify message transactions

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

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

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

5.2 Sync transaction

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

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

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

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

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

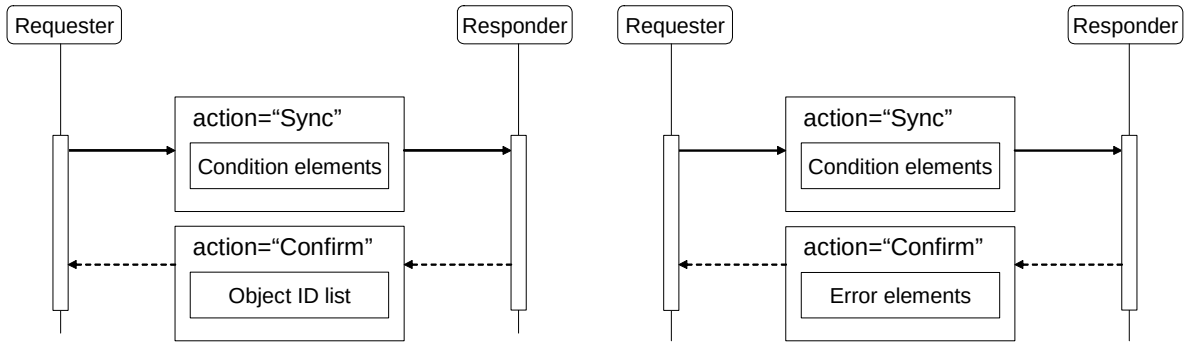


Figure 12 Sync message transaction

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

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

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

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

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

5.2.1 Sync message

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

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

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

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

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

```
</Condition>
</ProductionOrder>
```

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

```
<ProductionOrder id=”B-1” transaction=”03” action=”Confirm” event=”E01” sender=”B”>
<Order id=”1201”/>
<Order id=”1204”/>
<Order id=”1223”/>
</ProductionOrder>
```

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

5.2.2 Procedure of information owner

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

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

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

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

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

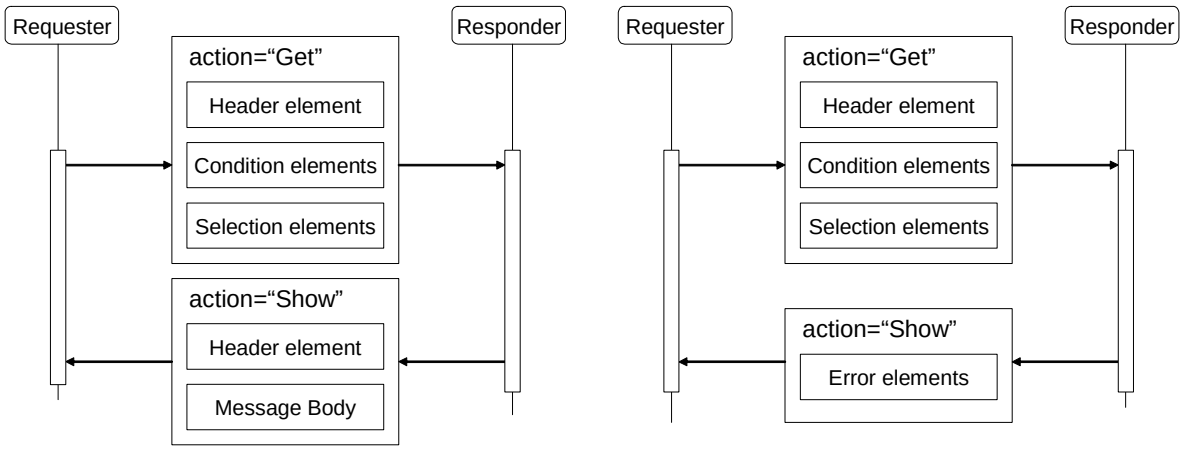
```
<ProductionOrder id=”B-2” transaction=”04” action=”Notify” event=”E01” sender=”B”>
<Order id=”1204”>
  <Produce><Qty value=”200”/></Produce>
</Order>
</ProductionRecord>
```

6 Information Query

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

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

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



(a) Normal case (b) Error case

Figure 13 Get -Show message transactions

6.1 Target domain objects

6.1.1 Selection by object IDs

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

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

```
<PartyRecord id="A-2" transaction="02" action="Get" sender="A">
<Condition id="0001"/>
<Condition id="0005"/>
<Condition id="0013"/>
<Selection type="all"/>
</PartyRecord>
```

6.1.2 Selection by Property elements

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

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

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

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

```
<ProductRecord id=”A-3” transaction=”03” action=”Get” sender=”A”>
<Condition>
  <Property name=”pps:color” value=”white” />
</Condition>
<Selection type=”all”/>
</ProductRecord>
```

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

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

```
<ProductRecord id=”A-4” transaction=”04” action=”Get” sender=”A”>
<Condition name=”pps:child” value=”A001”/>
<Selection type=”all”/>
</ProductRecord>
```

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

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

```
<ProductRecord id=”A-5” transaction=”05” action=”Get” action=”A”>
<Condition>
  <Property name=”pps:price”><Qty value=”2000” condition=”min”/></Property>
</Condition>
<Selection type=”all”/>
</ProductRecord>
```

6.1.3 Disjunctive and conjunctive conditions

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

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

```
<ProductRecord “A-6” transaction=”06” action=”Get” sender=”A”>
<Condition>
  <Property name=”pps:child” value=”A001”/>
  <Property name=”pps:child” value=”A002”/>
</Condition>
<Selection type=”all”/>
</ProductRecord>
```

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

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

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

6.1.4 Selection by wildcard

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

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

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

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

6.2 Target domain property

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

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

6.2.1 All available properties

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

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

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

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

6.2.2 Selecting domain property

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

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

```
<PartyRecord id=”A-10” transaction=”10” action=”Get” sender=”A”>
<Selection>
  <Property name=”pps:key”/>
  <Propertyv name=”pps:name” />
  <Property name=”pps:priority” />
</Selection>
</PartyRecord>
```

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

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

```
<LotRecord id=”A-11” transaction=”11” action=”Get” sender=”A”>
<Selection>
  <Property name=”user:calculation-1”
      path=”Spec[@type=’user:calculation-1’]/Qty/@value”/>
</Selection>
</LotRecord>
```

6.2.3 Sorting by property value

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

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

Example 19: Data request with sorting

```
<ProductRecord id=”A-12” transaction=”12” action=”Get” sender=”A”>
<Selection>
  <Property name=”pps:parent” sort=”asc”/>
  <Property name=”pps:name” sort=”asc”/>
</Selection>
</ProductRecord>
```

Example 20: An example of response of the previous example

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

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

6.2.4 Calculation of property value

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

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

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

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

Example 21: Requests to calculate summary of total price

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

Example 22: The corresponding response of the previous example

```
<CustomerOrder id="B-13" transaction="13" action="Show" sender="B">
<Header count="3">
  <Property name="pps:price" calc="sum"><Price value="2500"/></Property>
</Header>
<Order id="001" item="Product-1"><Price value="1000"/></Order>
<Order id="004" item="Product-1"><Price value="1000"/></Order>
<Order id="007" item="Product-1"><Price value="500"/></Order>
</CustomerOrder>
```

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

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

```
<CustomerOrder id="A-14" transaction="14" action="Get" sender="A">
<Selection>
  <Property calc="count"/>
</Selection>
</CustomerOrder>
```

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

```
<CustomerOrder id=”B-14” transaction=”14” action=”Show” sender=”B”>
<Header>
  <Property calc=”count”><Qty value=”55”/></Property>
</Header>
</CustomerOrder>
```

6.3 Multiple property

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

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

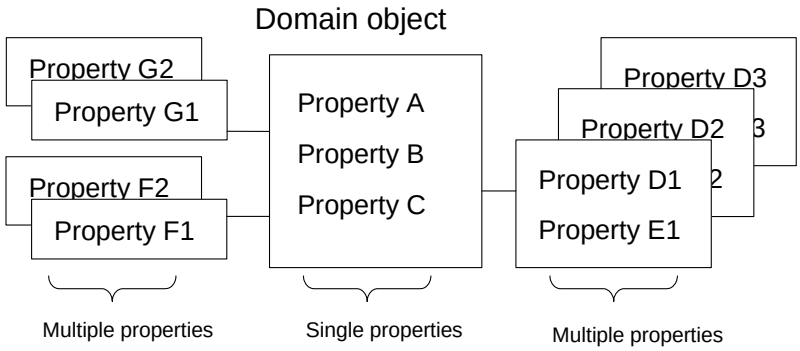


Figure 14 Single property and Multiple property

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

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

```
<PartyRecord id=”A-15” transaction=”15” action=”Get” sender=”A”>
<Condition id=”001”/>
<Selection>
  <Property name=”pps:id” />
  <Property name=”pps:name”/>
</Selection>
<Selection>
  <Property name=”pps:calendar-date” />
  <Property name=”pps:calendar-value”/>
  <Condition>
    <Property name=”pps:calendar-date”>
       <Time value=”2006-04-01T00:00:00” condition=”earliest”/>
    </Property>
    <Property name=”pps:calendar-date”>
      <Time value=”2006-05-01T00:00:00” condition=”latest” exclusive=”true”/>
    </Property>
  </Condition>
```

```
</Selection>
</PartyRecord>
```

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

```
<PartyRecord id="B-15" transaction="15" action="Show" sender="B">
<Party id="001">
  <Capacity status="pps:holiday"><Time value="2006-04-01T00:00:00"/></Capacity>
  <Capacity status="pps:work"><Time value="2006-04-02T00:00:00"/></Capacity>
  <Capacity status="pps:work"><Time value="2006-04-03T00:00:00"/></Capacity>
  …
  <Capacity status="pps:work"><Time value="2006-04-30T00:00:00"/></Capacity>
</Party>
</PartyRecord>
```

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

6.4 Using Header element

6.4.1 Query by header element

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

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

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

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

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

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

Example 26: An answer of the previous message

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

6.4.2 Count of domain objects

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

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

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

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

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

```
<CustomerOrder id=”A-17” transaction=”17” action=”Get” sender=”A”>
<Selection offset=”100” count=”10”/>
  <Property name=”pps:id” sort=”desc”/>
</Selection>
</CustomerOrder>
```

6.5 Show message

6.5.1 Structure of Show message

Show message has the same stricture as the structure of Notify message. This message SHOULD have a value of “Show” at the *action* attribute.

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

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

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

```
<CustomerOrder id=”B-18” transaction=”18” action=”Show” sender=”B”>
<Header id=”001” count=”3” title=”OrderSheet”>
  <Property name=”pps:party” value=”K-Inc.” display=”CSTM”/>
  <Property type=”selection” name=”pps:id” display=”PN”/>
  <Property type=”selection” name=”pps:name” display=”NAME”/>
  <Property type=”selection” name=”pps:qty” display=”QTY”/>
  <Property type=”selection” calc=”sum” name=”pps:price” display=”PRICE”>
    <Qty value=”1200”/></Property>
</Header>
<Order id=”001-1” item=”Product-A1”><Qty value=”1”/></Order>
<Order id=”001-2” item=”Product-A2”><Qty value=”10”/></Order>
<Order id=”001-3” item=”Product-A3”><Qty value=”3”/></Order>
</CustomerOrder>
```

6.5.2 Header in Show message

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

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

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

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

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

```
<ProductRecord id=”A-19” transaction=”19” action=”Get” sender=”A”>
<Condition>
  <Property name=”pps:parent” vaue=”A001”/>
</Condition>
<Selection>
  <Property name=”pps:id”/>
  <Property name=”pps:name”/>
</Selection>
<Header title=”BillOfMaterials” id=”A001” >
  <Property name=”pps:name”/>
  <Property name=”pps:price”/>
  <Property name=”pps:price-unit”/>
</Header>
</ProductRecord>
```

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

```
<ProductRecord id=”B-19” transaction=”19” action=”Show” sender=”B”>
<Header title=”BillOfMaterials” id=”A001” count=”3”>
  <Property name=”pps:name” value=”Product A001”/>
  <Property name=”pps:price”><Qty value=”2000”/></Property>
  <Property name=”pps:price-unit” vaule=”yen”/>
  <Property type=”condition” name=”pps:parent” vaue=”A001”/>
  <Property type=”selection” name=”pps:id”/>
  <Property type=”selection” name=”pps:name”/>
</Header>
<Item id=”A001-01” name=”Part A001-01”/>
<Item id=”A001-02” name=”Part A001-02”/>
<Item id=”A001-03” name=”Part A001-03”/>
</ProductRecord>
```

7 XML Elements

7.1 Error element

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

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

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

```
<xsd:element name="Error">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="ref" type="xsd:string"/>
    <xsd:attribute name="code" type="xsd:string"/>
    <xsd:attribute name="location" type="xsd:string"/>
    <xsd:attribute name="status" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *id* attribute SHOULD represent identifier that application can identify the error data.
- *ref* attribute SHOULD represent the transaction message id that has the errors.
- *code* attribute SHOULD represent unique identifier of the error categories. The error code SHOULD consist of three digits. If the first digit is 0, then the code SHOULD represent as follows:
  - “000” represents “Unknown error”.
  - “001” represents “Connection error”.
  - “002” represents “Authorization error”.
  - “003” represents “Application is not ready”.
  - “004” represents “Message buffer is full”.
  - “005” represents “Syntax error (communication)”.
  - “006” represents “Syntax error (application logic)”.
  - “007” represents “Requested task is not supported”.
  - “008” represents “Requested task is denied”.
  - “009” represents “No data object requested in the message”.
  - “010” represents “Data object requested already exists”.
  - “011” represents “Application error”.
  - “012” represents “Abnormal exception”.
- *location* attribute SHOULD represent the location of error texts.
- *status* attribute SHOULD represent a status. Values of this attribute SHOULD include:
  - “error” represents that the message is error notification.
  - “warning” represents that the message is warning.
- *description* attribute SHOULD represent any description of the error explanations.

7.2 App element

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

This element SHOULD be consistent with the following XML schema.

```
<xsd:element name="App">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

7.3 Condition element

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

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

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

```
<xsd:element name="Condition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="wildcard" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string"/>
    <xsd:attribute name="version" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

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

- *id* attribute SHOULD represent the identifier of the target domain object. When the target object is known, then this value is specified instead of describing any other conditions.
- *wildcard* attribute SHOULD represent the name of property that is used to apply wildcard value. The wildcard text is specified in the *value* attribute.
- *value* attribute SHOULD represent the wildcard text for selecting the target domain objects. The text is interpreted by regular expression rules.
- *version* attribute SHOULD represent version name of the target object. The format of version texts is managed in application programs. Values of this attribute MAY include:
  - “latest” --- the latest version object
  - “earliest” – the earliest version object
  - any string that represent a version identifier

7.4 Selection element

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

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

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

```
<xsd:element name="Selection">
<xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="Condition" maxOccurs="unbounded"/>
   <xsd:element ref="Property" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="multi" type="xsd:int"/>
  <xsd:attribute name="count" type="xsd:int"/>
  <xsd:attribute name="offset" type="xsd:int"/>
</xsd:complexType>
</xsd:element>
```

- *Condition* element SHOULD represent any condition for selecting members of a multiple property, when the *multiple* attribute is greater than 1. Change or Get message can restrict its target by this condition.
- *Property* element SHOULD represent any property required to describe in the target domain objects. In the case of Get-Show model, the corresponding information of this property is addressed in the body of the response message.

- *type* attribute SHOULD represent the type of action after selecting the target properties. The available values are defined depending on the type of message.
  - "insert" for Change message represents that the property value is inserted, this is default value,
  - "update" for Change message represents that the property value is updated,
  - "delete" for Change message represents that the property value is deleted,
  - "none" for Get message represents that the target is specified by *Property* element, this is default value,
  - "typical" for Get message represents that the target property is typical set,
  - "all" for Get message represents that the target property is all properties in the object.
- *multi* attribute SHOULD show whether the selected property is multiple or single one. When the property is single, the value of "type" attribute in Change message is not necessary because the operation is always updating.
- *count* attribute
- *offset* attribute

7.5 Header element

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

element MAY be used to make brief query of domain object that is not in the target of domain document. The *Header* element SHOULD be specified in the transaction element.

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

```
<xsd:element name="Header">
<xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string"/>
  <xsd:attribute name="class" type="xsd:string"/>
  <xsd:attribute name="title" type="xsd:string"/>
  <xsd:attribute name="count" type="xsd:int"/>
  <xsd:attribute name="offset" type="xsd:int"/>
</xsd:complexType>
</xsd:element>
```

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

- *id* attribute SHOULD represent ID of the target object that is shown in the header by representing its property in the “Property” element.
- *class* attribute SHOULD represent the target domain object that the header shows the information in *Property* elements. Default value is the name of domain object the document refers as default.
- *title* attribute SHOULD represent a title of the message document.
- *count* attribute SHOULD represent the number of domain objects in the message. When this attribute is used in Confirm message and Show message, the value equals to the number of object in the body of message. In Get message, the value represents the maximum number the receiver is expecting the objects in the Show message.
- *offset* attribute SHOULD represent the offset number of data list. When the objects in the message is not all of the existing objects in the sender, the offset value shows the relative position of the first object on the message body in the whole objects. This attribute can be used in Get message as a request to offset the response data.

7.6 Property element

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

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

```
<xsd:element name="Property">
 <xsd:complexType>
  <xsd:choice>
   <xsd:element ref="Qty" minOccurs="0"/>
   <xsd:element ref="Char" minOccurs="0"/>
   <xsd:element ref="Duration" minOccurs="0"/>
   <xsd:element ref="Time" minOccurs="0"/>
  </xsd:choice>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="name" type="xsd:string"/>
```

```
  <xsd:attribute name="path" type="xsd:string"/>
  <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*, *Duration*, and *Time* elements SHOULD represent the value of the primitive. If the data type of the value is string and there is no additional constraint, the *value* attribute can be used instead of the *Char* element.

- *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. The data type of this attribute is string, so other data such as integer and dateTime SHOULD be specified in the child element. When the property is under *Condition* element or *Selection* element, this shows a constraint to be equivalent.
- *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.

A. Implementation level

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

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

The following table prescribes the implementation levels.

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

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

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 *Selection* elements. A *Selection* element MAY have *Condition* elements. Change message MAY have *Selection* element that has *Condition* elements. Level 1 Implementation: Get message SHOULD NOT have more than one *Selection* elements. A *Selection* element SHOULD NOT have *Condition* 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: *Property* element MAY have *sort* attribute and *calc* attribute to request sorting and calculation of domain objects in the Show message. Level 1 Implementation: *Property* 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 *Header* element (see 6.4.1)	Full Functions: Get message MAY have *Header* 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 *Header* element. Query by header object is not available. Restriction of domain object in Show message is not available.

1271

1272

B. Acknowledgements

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

Participants:

Shinya Matsukawa, Hitachi
Tomohiko Maeda, Fujitsu
Masahiro Mizutani, Unisys Corporation
Akihiro Kawauchi, Individual Member
Yuto Banba, PSLX Forum
Osamu Sugi, PSLX Forum
Hideichi Okamune, PSLX Forum
Hiroshi Kojima, PSLX Forum
Ken Nakayama, Hitachi
Yukio Hamaguchi, Hitachi
Tomoichi Sato, Individual
Hiroaki Sasaki, Individual

C. Revision History

Revision	Date	Editor	Changes Made