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

Public Review Draft 02

24 Oct 2009

Specification URIs:

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

Previous Version:

http://docs.oasis-open.org/pps/v1.0/cs01/pps-transaction-messages-1.0.doc
http://docs.oasis-open.org/pps/v1.0/cs01/pps-transaction-messages-1.0.html
http://docs.oasis-open.org/pps/v1.0/cs01/pps-transaction-messages-1.0.pdf

Latest Version:

http://docs.oasis-open.org/pps/v1.0/pr02/pps-transaction-messages-1.0.doc
http://docs.oasis-open.org/pps/v1.0/pr02/pps-transaction-messages-1.0.html
http://docs.oasis-open.org/pps/v1.0/pr02/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/2009

Abstract:

OASIS PPS (Production Planning and Scheduling) specifications deal with problems of decision-making in all manufacturing companies who want to have a sophisticated information system for production planning and scheduling. PPS specifications provide XML schema and communication protocols for information exchange among manufacturing application programs in the web-services environment. Part 3: Transaction Messages especially focuses on transaction messages that represent domain information sending or receiving by application programs 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", 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

Figures

1 Introduction

This part of PPS specifications provides structure and rules of XML transaction elements for messaging between two application programs. Core part of XML representations of the messages consist of XML core elements that are defined in [PPS01]. This specification 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 has message documents which are sent or received as a message. This part does not define type of document, but defines a data structure of message elements, transaction elements and document element that may be created for any particular circumstances. Each document element has domain objects in the production planning and scheduling domain. The domain objects can be represented by nine primitive elements defined in [PPS01].

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

[PCRE] PCRE(Perl Compatible Regular Expression), http://www.pcre.org/

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-schema-1.0.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 to the normative text of this specification.

1.5 Terms and definitions

Application profile

Collections of profile specifications for all application programs that may be involved in the communication group who exchanges PPS messages. This information is defined by platform designer to provide all available domain documents, domain objects and domain properties.

Domain document

Document that is a content of message sent or received between application programs, and is processed by a transaction. Domain document consists of a verb part and a noun part. Verbs such as add, change and remove affect the types of messages, while nouns represented by domain objects show the classes of domain objects. Specific classes of domain documents can be defined by platform designer to share the domain information.

Domain object

Object necessary for representing production planning and scheduling information in manufacturing operations management. Domain objects are contents of a domain document, and represented by primitive elements. Specific classes of domain objects can be defined by platform designer to share the domain information.

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. Specific properties of domain objects can be defined by platform designer to share the domain information, and additionally defined by each application designer.

Implementation profile

Specification of capability of an application program in terms of exchanging PPS messages. The profile includes a list of available documents and their properties that may be exchanged in PPS messages among production planning and scheduling applications.

Messaging model

Simple patterns of messaging between sender and receiver, or requester and responder. Four message models: NOTIFY, PUSH, PULL, SYNC are defined from an application independent perspective.

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 transaction to process message documents which is sent or received between application programs. Transaction element can control a transaction process of application program database by commitment and rollback. Transaction element may request confirmation from receiver if the message has been received properly.

2 Messaging model

2.1 Basic Unit of messaging

Two basic unit of messaging are defined in this specification. 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 responding message is mandatory or optional depending on the service. The receiver or responder may be multiple at one transaction, so as to make broad casting.

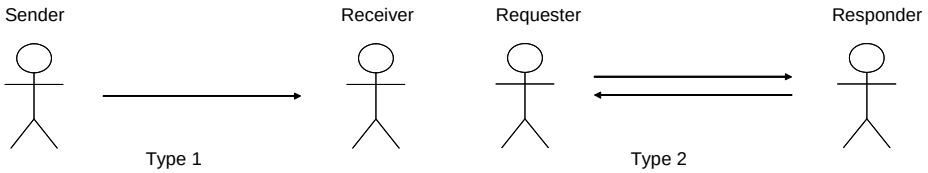


Figure 1 Basic unit of messaging

The basic units used to define several messaging models in later sections. However 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 a building block. 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 the simple messaging unit, considering security, reliability, efficiency and so forth. In such cases, messages may be sent several times for the one arrow in Figure 1. This is also not in the scope of this part.

Application programs communicate using the basic unit of messaging to perform particular business logics. One or more than one transactions of domain documents are contained in each message.

2.2 Message classes

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

Table 1 Action classes of domain documents

Action 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 responds from the responder to the requester as a confirmation of the results.
Notify	The message informs any domain objects to the receiver as a 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 responding Show message.
Show	The message responses the requested information of domain objects to the Get message from the requester.

In order to ask the confirmation from responders, domain documents that perform with Add, Change, Remove or Sync action 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 NOTIFY model (Type 1)

Basic massaging unit of Type 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 to make a task for the message.

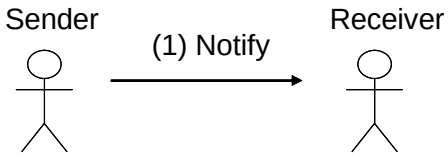


Figure 2 NOTIFY model

2.3.2 PUSH model (Type 2)

In PUSH model, domain document with Add action, Change action and Remove action can be requested and processed by applications. This model is enabled by type 1 messaging unit.

In Add transaction, 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.

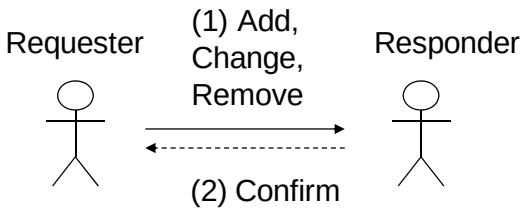


Figure 3 PULL model

Change transaction 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.

Remove transaction 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.

Responder processes the requested actions, and if necessary, responds confirmation documents to the requester.

2.3.3 PULL model (Type 2)

PULL model is defined for one or more than one actions of Get-Show transactions. 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 which is managed by the responder.

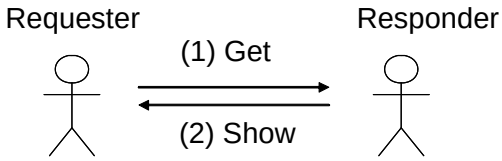


Figure 4 PULL model

2.3.4 SYNC model (Type 2 and Type 1)

SYNC model consists of a Sync transaction (Type 2) and several Notify transactions (Type 1). 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 in order to send Notify messages by invoking another Notify transaction. Notify messages are sent repetitively when the event occurs until the Sync request is canceled.

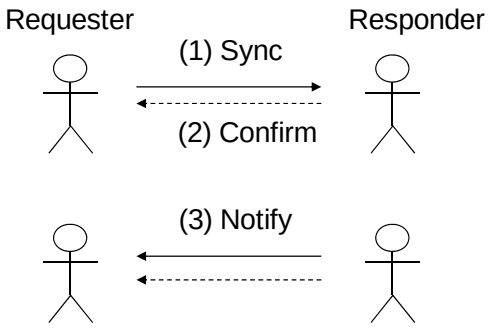


Figure 5 SYNC model

2.4 Procedures on responders

2.4.1 Common tasks

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

- The responder, who receives a proper Get document, 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 document, 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 document, 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 document, 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 document SHOULD have capability to make the following tasks when the message received has a confirmation request.

- The responder SHOULD send a Confirm document to the requester when the Add document received has an attribute of confirm="Always". The Confirm document SHOULD have either error information or the id list that shows all the objects added to the database.
- The responder SHOULD send a Confirm document to the requester when the Change document received has an attribute of confirm="Always". The Confirm document SHOULD have either error information or the id list that shows all the objects changed in the database.
- The responder SHOULD send a Confirm document to the requester when the Remove document received has an attribute of confirm="Always". The Confirm document SHOULD have either error information or the id list that shows all the objects deleted in the database.
- The responder SHOULD send a Confirm document to the requester when the Sync document received has an attribute of confirm="Always". The Confirm document SHOULD have either error information or the id list that shows all the objects to be monitored for synchronization.
- The responder SHOULD NOT send a Confirm document to the requester when the document received has an attribute of confirm="Never".

2.4.3 Error handling

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

- In PULL model, responder, who receives a Get document and is hard to respond in normal processes because of errors, SHOULD send a Show document with the error information to the requester.
- In PUSH model and SYNC model, responder who receives a document that has attribute of confirm="OnError" or "Allways" and detects errors during the process requested SHOULD send a Confirm document with the error information to the requester.
- The responder SHOULD NOT send a Confirm document nor Show document to the requester when the document received has an attribute of confirm="Never", even if there is an error.

3 Add, Change and Remove (PUSH model)

3.1 Add transaction

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

When the Add document request to add domain objects with ID specified at the “id” attribute, responder SHOULD check existence of the ID in its database and add the data if the corresponding data does not already exist in the database. If the document has an ID that already exists in the database, the responder SHOULD NOT add the data.

When the Add document 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.

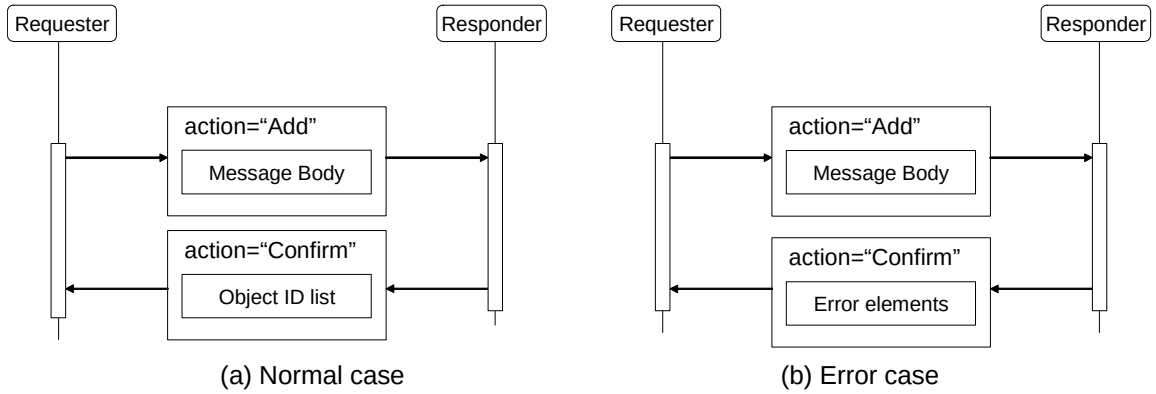


Figure 6 Add transactions

Example: Document to add three Product Records

```
<Document id=”A-1” name=”Product” action=”Add”>
<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>
</Document>
```

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

Example: Add document using a *Condition* element

```
<Document id=”A-2” name=”Product” action=”Add” >
<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”/>
</Document>
```

The response to Add document can be done by sending a Confirm document 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 document as a result of the previous Add document.

Example: Confirm document as a response of an Add transaction

```
<Document id=”B-1” name=”Product” action=”Confirm” >
<Item id=”001” />
<Item id=”002” />
<Item id=”003” />
</Document>
```

3.2 Change transaction

Change document 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 or more than one 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 this Change process.

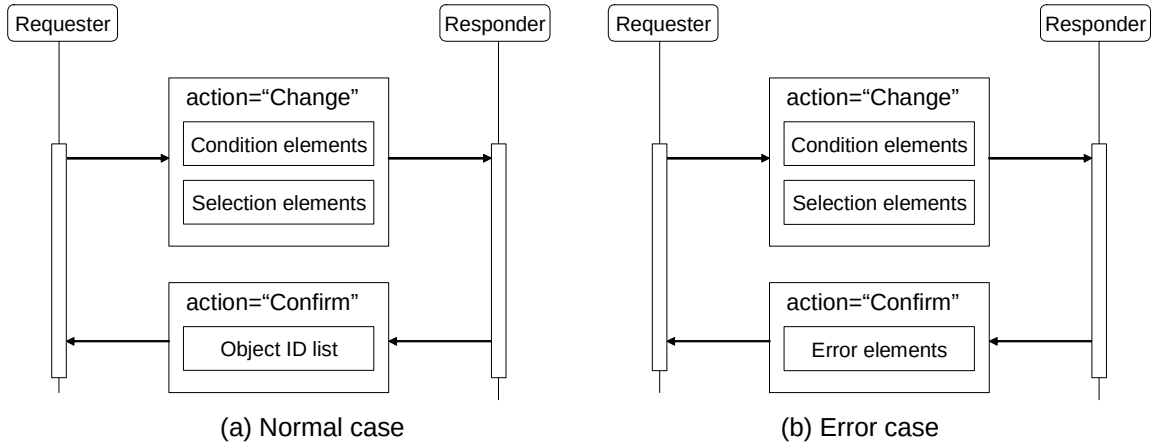


Figure 7 Change transactions

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 document.

3.2.1 Insert property (Level 2 function)

For the multiple primitives that have the same property name in the same object, an insert property document 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: Add information of new level 10 as the latest stock value.

```
<Document id=”A-4” name=”Product” action=”Change” >
<Condition id=”001”/>
<Selection type=”Insert” >
  <Property name=”pps:stock”><Qty value=”10”/></Property>
</Selection>
</Document>
```

3.2.2 Update property (Level 2 function)

When the value of *type* attribute of *Selection* element is “Update”, the properties in the *Selection* element are for updating the current properties in the owner’s database. 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 selected 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: Document requests to change the usage of A001-2 from 1 to 4.

```
<Document id=”A-5” name=”Product” action=”Change” >
<Condition id=”A001”/>
<Selection type=”Update” >
  <Condition><Property name=”pps:child”><Char value=”A001-2”/></Property></Condition>
  <Property name=”pps:child-value”><Qty value=”4”/></Property>
</Selection>
</Document>
```

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

```
<Dosument name=”Item” id=”A001”>
<Compose type=”pps:child” item=”A001-1”><Qty value=”1”/></Compose>
<Compose type=”pps:child” item=”A001-2”><Qty value=”1”/></Compose>
<Compose type=”pps:child” item=”A001-3”><Qty value=”1”/></Compose>
</Dosument>
```

Example: Revised status of the product data

```
<Dosument name=”Item” id=”A001”>
<Compose type=”pps:child” item=”A001-1”><Qty value=”1”/></Compose>
<Compose type=”pps:child” item=”A001-2”><Qty value=”4”/></Compose>
<Compose type=”pps:child” item=”A001-3”><Qty value=”1”/></Compose>
</Document>
```

3.2.3 Delete property (Level 2 function)

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: Usage of “Machine-1” by the process “Proc-1” is requested to delete.

```
<Document id=”A-6” name=”ProductionProcess” action=”Change” >
<Conditoin id=”Proc-01”/>
<Selection type=”Delete”>
   <Condition><Property name=”pps:equipment”><Char value=”Machine-1”/></Property></Condition>
</Selection>
```

```
</Document>
```

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

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

3.3 Remove transaction

Remove document 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 an error message, unless the confirm attribute is “Never”. Removing objects means that the data in the owner’s 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 the target domain objects.

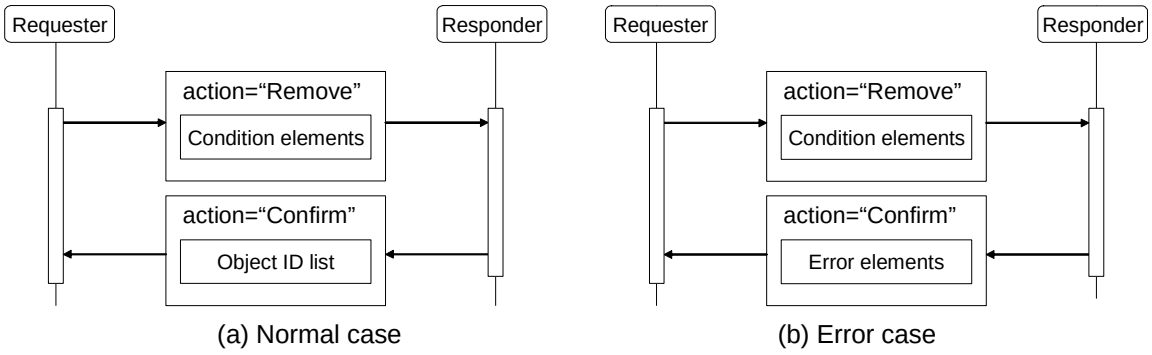


Figure 8 Remove transactions

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

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

4 Notify and Sync (NOTIFY and SYNC model)

4.1 Notify transaction

Notify document SHOULD have a value of “Notify” in the *action* attribute. The figure shows that transaction pattern of Notify document exchange. The sender of Notify document will not receive its response from the receiver.

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

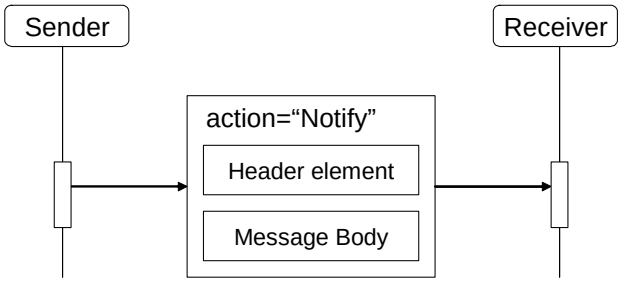


Figure 9 Notify transactions

Notify document 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 described in the body of a Notify document.

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

```
<Document id=”A-9” name=”SalesOrder” action=”Notify” >
<Header id=”001” count=”3” title=”Order Form”>
  <Property type=”Target” name=”pps:party” display=”C-Name”><Char value=”K-Inc.”/></Property>
  <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”><Spec type=”pps:plan”><Qty value=”1”/></Spec></Order>
<Order id=”001-2” item=”Product-A2”><Spec type=”pps:plan”><Qty value=”10”/></Spec></Order>
<Order id=”001-3” item=”Product-A3”><Spec type=”pps:plan”><Qty value=”3”/></Spec></Order>
</Document>
```

4.2 Synchronizing proess

In order to synchronize information of users with the information of the owner’s database, 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 document is used to establish a relationship of synchronization by requesting subscription of the event occurrence detected by the information owner.

When a synchronization request specified using a Sync document is accepted by responder, e.g., the information owner, the responder SHOULD be ready to send a notification document by invoking another transaction when the corresponding event occurs. The notification documents are not included in the Sync transaction. Notification of change of the property value will be invoked as a different transaction independent from the Sync transaction.

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

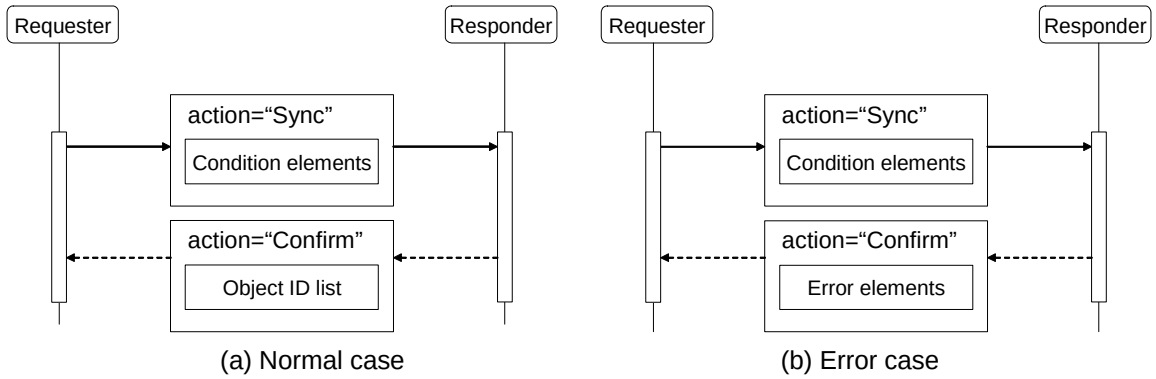


Figure 10 Sync transaction

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

According to the implementation profile specification format, 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 before 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 that is 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.

4.2.1 Sync document

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

Sync document MAY specify particular domain objects that have been managed by the responder at the time and is possible to monitor to detect the event. *Condition* element allows the requester to make request of synchronization for several domain objects by sending one Sync document.

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

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

```
<Document id="A-3" name="ProductionOrder" action="Sync" event="E01" >
<Condition>
  <Property name="pps:item"><Char vaue="A001"/></Property>
</Condition>
</Document>
```

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

```
<Document id="B-1" name="ProductionOrder" action="Confirm" event="E01" >
<Order id="1201"/>
<Order id="1204"/>
<Order id="1223"/>
</Document>
```

Once a *Sync* document is received without error, the synchronization request becomes effective until the responder will get a cancel request of the subscription, or the responder will stop the event detection process. In order to cancel the *Sync* request by requester, the requester SHOULD send a *Sync* document under a *Transaction* element that has *type* attribute with "Cancel" vale. When the responder receives cancelation of the *Sync* transaction, the responder SHOULD cancel the synchronization request corresponding to the transaction id. If the cancel request has new transaction id, then all transactions restricted by the specified event name and *Condition* element are canceled.

4.2.2 Procedure of information owner

Information owner, who has a capability of event monitoring and 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 document 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 process to a middle-ware information broker.

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

Example: Notify of event "E01" that shows a change of "production result" of production orders.

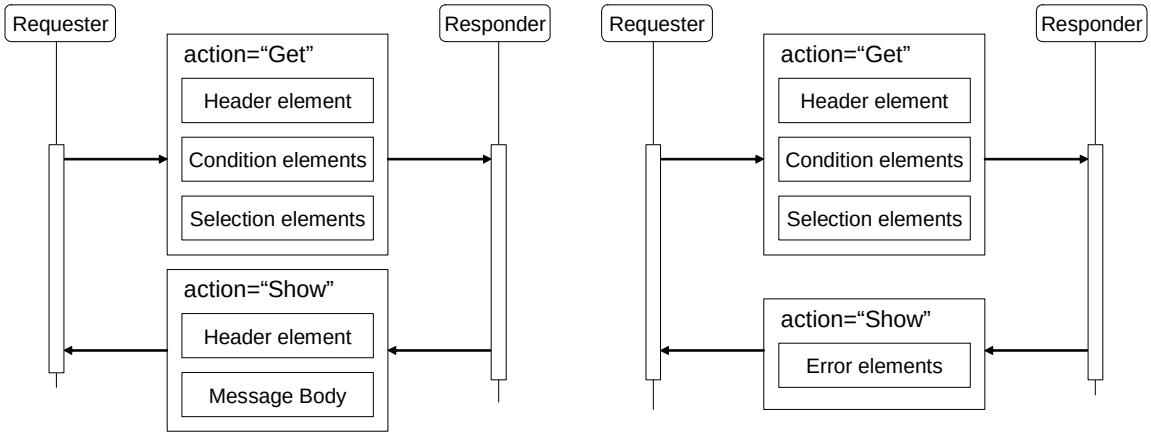
```
<Document id="B-2" name="ProductionOrder" action="Notify" event="E01" >
<Order id="1204">
  <Produce><Qty value="200"/></Produce>
</Order>
</Document>
```

5 Information Query (PULL model)

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

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

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



(a) Normal case (b) Error case

Figure 11 Get -Show transactions

5.1 Target domain objects

5.1.1 Selection by object IDs

The simplest way to select domain objects is describing 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: Three objects that have "0001", "0005", "0013" as ID are requested.

```
<Document id="A-2" name="Customer" action="Get" >
<Condition id="0001"/>
<Condition id="0005"/>
<Condition id="0013"/>
<Selection type="All"/>
</Document>
```

5.1.2 Selection by Property elements

The second way to select domain objects is to specify *Property* elements in the *Condition* element under the *Document* 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, then the domain object SHOULD be selected.

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

```
<Document id=”A-3” name=”Product” action=”Get” >
<Condition>
  <Property name=”pps:color”><Char value=”white” /></Property>
</Condition>
<Selection type=”All”/>
</Document>
```

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

```
<Document id=”A-4” name=”Product” action=”Get” >
<Condition>
  <Property name=”pps:child”><Char value=”A001”/></Property>
</Condition>
<Selection type=”All”/>
</Document>
```

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*, and *Time* element that has *condition* attribute. Available types of condition SHOULD include GE (greater than or equal), LE (less than or equal), GT (greater than), LT (less than), EQ (equal), NE (not equal).

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

```
<Document id=”A-5” name=”Product” action=”Get” >
<Condition>
  <Property name=”pps:price”><Qty value=”2000” condition=”GE”/></Prperty>
</Condition>
<Selection type=”All”/>
</Document>
```

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

```
<Document “A-6” name=”Product” action=”Get” >
<Condition>
  <Property name=”pps:child”><Char value=”A001”/></Property>
  <Property name=”pps:child”><Char value=”A002”/></Property>
</Condition>
```

```
<Selection type="All"/>
</Document>
```

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

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

```
<Document id="A-7" name="Product" action="Get" >
<Condition><Property name="pps:child"><Char value="A001"/></Property></Condition>
<Condition><Property name="pps:child"><Char value="A002"/></Property></Condition>
<Selection type="All"/>
</Document>
```

5.1.4 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 [PCRE] are applied for interpreting the wildcard string.

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

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

```
<Document id="A-8" name="SalesOrder" action="Get" >
<Condition wildcard="pps:delivery" value="Boston"/>
<Selection type="All"/>
</Document>
```

5.2 Target domain property

When the target domain objects are determined, *Get* document 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 *Selection* elements are included and corresponding values are described by the responder in the *Show* document.

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

5.2.1 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* document. The list of properties to return is decided by the responder.

When value "Typical" is described 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 according to the profile [PPS03].

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

```
<Document id="A-9" name="ResourceCapacity" action="Get" >
<Selection type="All"/>
</Document>
```

5.2.2 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 described in the *name* attribute of *Property* element in the *Get* document. Property name is defined in the application profile or the implementation profile.

Example: The objects in the responding document are required with properties of key, name and priority.

```
<Document id="A-10" name="Party" action="Get" >
<Selection>
  <Property name="pps:key"/>
  <Property name="pps:name" />
  <Property name="pps:priority" />
</Selection>
</Document>
```

When the property required has not been defined in the profile, Get document MAY request user-made properties by specifying its own texts following the prefix of "user:".

5.2.3 Sorting by property value (Level 2 function)

Sorting request of the domain objects in the Show document can be described 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 document SHOULD sort the domain objects by descending or ascending order, respectively.

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

Example: Data request with sorting

```
<Document id="A-12" name="Product" action="Get" >
<Selection>
  <Property name="pps:parent" sort="Asc"/>
  <Property name="pps:name" sort="Asc"/>
<Selection>
</Document>
```

Example: An example of response of the previous example

```
<Document id="B-12" name="Product" action="Show" >
<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>
</Document>
```

5.2.4 Calculation of property value (Level 2 function)

Property element in a *Selection* element MAY represent a request of calculation of property values that are selected by the *Get* document. 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 function.

The name of property that should be calculated MAY be described in *name* attribute of the *Property* element. Then, the values of the property SHOULD be calculated using the function describing at the *calc* attribute.

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

The responder who receives *Get* document SHOULD answer by calculating the target property value, and describes it at the corresponding *value* attribute of Qty, Char and Time element in the Property element depending on the data type.

Example: Requests to calculate summary of total price

```
<Document id="A-13" name="SalesOrder" action="Get" >
<Selection>
  <Property name="pps:price" calc="Sum"/>
</Selection>
<Selection type="All"/>
</Document>
```

Example: The corresponding response of the previous example

```
<Dosument name="SalesOrder" id="B-13" action="Show" >
<Header count="3">
  <Property name="pps:price" calc="Sum"><Qty value="2500"/></Property>
</Header>
<Order id="001" item="Product-1"><Price><Qty value="1000" unit="USD"/></Price></Order>
<Order id="004" item="Product-1"><Price><Qty value="1000" unit="USD"/></Price></Order>
<Order id="007" item="Product-1"><Price><Qty value="500" unit="USD"/></Price></Order>
</Dosument>
```

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* document. In the case that "Count" value is specified in *calc* attribute, name attribute of *Property* element MAY NOT be specified.

Example: Request of counting the number of data

```
<Document id="A-14" name="SalesOrder" action="Get" >
<Selection>
  <Property calc="Count"/>
</Selection>
</Document>
```

Example: The answer of the request of counting the data

```
<Document id="B-14" name="SalesOrder" action="Show" >
<Header>
  <Property calc="Count"><Qty value="55"/></Property>
</Header>
</Document>
```

This value is similar to the value of *count* attribute in *Header* element. The value described in the count attribute represents the actual number of objects in the document, whereas the value in Property element shows the actual number in the database managed by the responder.

5.3 Multiple property (Level 2 function)

A *Document* element for a simple *Get* transaction has one *Selection* element which has several properties required by the sender. 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 properties needs *Condition* element as its child element to represent conditions to select the instances.

From a modeling perspective, a multiple property can be defined by attribute objects which are associated with or contained by the target domain object. The target domain object and attribute objects has one-to-many relations. Figure 12 shows that Property A, B, and C is a single property, while Property D to G are multiple properties. In this figure, it is important that Property D and E are on the same attribute object, and then any conditions for those two properties are applied in the same manner to select satisfied attribute objects.

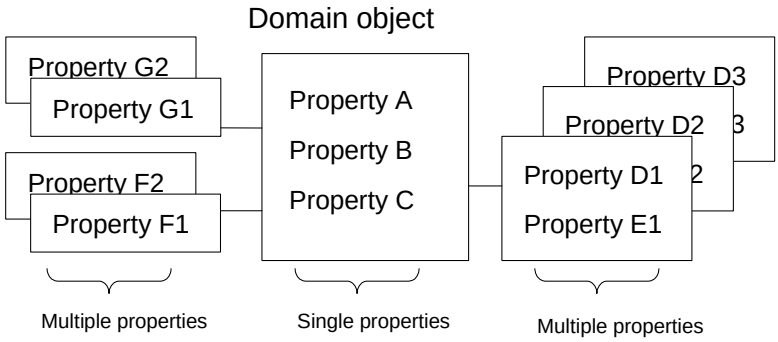


Figure 12: Single property and Multiple property

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. In the three *Selection* elements, one for the multiple properties has information of Property D and Property E at the same *Selection* element.

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

```
<Document id=”A-15” name=”Customer” action=”Get” >
<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=”GE”/>
    </Property>
    <Property name=”pps:calendar-date”>
      <Time value=”2006-05-01T00:00:00” condition=”LT”/>
    </Property>
  </Condition>
</Selection>
</Document>
```

Example: One possible answer to the previous document.

```
<Document id=”B-15” name=”Customer” action=”Show” >
```

```
<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>
</Document>
```

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.

5.4 Using Header element

5.4.1 Inquiry by header element (Level 2 function)

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

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

The target object selected in this brief inquiry is basically in the same class of the domain objects, unless the *class* attribute of *Header* element has another name of domain object. When the *class* attribute is described with a name of another domain object, the corresponding information of the domain objects will be answered in the *Header* of the Show document.

Multiple property MAY not be processed 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 selected and described in the answer document.

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

```
<Document id="A-16" name="Product" action="Get"
<Header id="001">
 <Property type="Target" name="pps:name"/>
</Header>
</Document>
```

Example: An answer of the previous document

```
<Document id="B-16" name="Product" action="Show" >
<Header id="001">
 <Property type="Target" name="pps:name"><Char value="Product-A"/></Property>
</Header>
</Document>
```

5.4.2 Count of domain objects (Level 2 function)

In Get document, *count* attribute of *Selection* element SHOULD represent the maximum number of objects described in the response message. If the value of the *count* attribute is 1 or more than 1, then the number described 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 documents. In such case, *offset* attribute of *Selection* element SHOULD be described as an offset number to skip the first objects while retrieving 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 addressed in the *Document* element. In the corresponding Show document, the attribute of *count* and *offset* are specified in the *Header* element instead of *Selection* element.

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

```
<Document id=”A-17” name=”SalesOrder” action=”Get” >
<Selection offset=”100” count=”10”/>
  <Property name=”pps:id” sort=”Desc”/>
</Selection>
</Document>
```

5.5 Show document

5.5.1 Structure of Show document

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

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

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

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

```
<Document id=”B-18” name=”SalesOrder” action=”Show” >
<Header id=”001” count=”3” title=”OrderSheet”>
  <Property name=”pps:party” display=”CSTM”><Char value=”K-Inc.”/></Property>
  <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>
</Document>
```

5.5.2 Header in Show document

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

Property elements described in the *Header* element consist of three types. First type is for properties of a header domain object requested by the Get document as a result of brief inquiry. All *Property* elements of this group SHOULD have a value “Target” at the *type* attribute or the attribute is not described. 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 document has the same value described in the property. Application program who responses the Show document MAY describe the properties simply by duplicating the corresponding *Property* elements in *Condition* element in the Get document, because the property to be described can be regarded as a condition of the domain objects.

The final group of properties comes from the *Selection* element of the Get document. 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 document. If the *Selection* element in the Get document requests calculation, results are described in the *value* attribute of *Qty*, Char or *Time* sub-element of the *Property* element. In addition, a value of *display* attribute MAY be described for any texts in the header area for printing on a formatted sheet.

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

```
<Document id=”A-19” name=”Product” action=”Get”>
<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>
</Document>
```

Example: The response to the previous Get document.

```
<Document id=”B-19” name=”Product” action=”Show”>
<Header title=”BillOfMaterials” id=”A001” count=”3”>
  <Property name=”pps:name”><Char value=”Product A001”/></Property>
  <Property name=”pps:price”><Qty value=”2000”/></Property>
  <Property name=”pps:price-unit”><Char vaule=”yen”/></Property>
  <Property type=”Condition” name=”pps:parent”><Char vaue=”A001”/></Property>
  <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”/>
</Document>
```

6 XML Elements

6.1 Message Structure

Message is defined as unit information to send or receive by an application program at one time. A message that is exchanged between two parties SHOULD consist of one or more transaction elements or an implementation profile.

The message content corresponds to any content in actual communication protocol such as SOAP, FTP and SMTP. Since this specification 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.

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:complexType name="MessageType">
  <xsd:choice>
    <xsd:element ref="ImplementProfile"/>
    <xsd:element ref="Transaction" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <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 message SHOULD have a unique id in the scope of the sender or the requester.
- *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 message is created.
- *description* attribute SHOULD represent any comments or descriptions.

Elements under this messageType element SHOULD follow the sentences:

- *ImplementProfile* element SHOULD represent a request of implementation profile or answer of implementation profile defined in [PPS03].
- *Transaction* element SHOULD represent transaction information to process in the responder.

In the case of representing XML format in messaging, the name of XML element can be described according to the following XML schema. In the case of describing in specific protocols such as SOAP, the payload body SHOULD be defined using MessageType.

```
<xsd:element name="Message" type="MessageType"/>
```

6.2 Transaction element

A transaction element represents information of a transaction step. In the case where application need to commit several actions during transaction, and where it need to cancel and rollback the actions it has already processed, transaction element can control such operations.

Transaction element SHOULD consist of zero or more than zero domain documents. When it has multiple documents, the first document in the content is the primary document in the transaction.

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="Transaction">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Document" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="confirm" type="xsd:string"/>
    <xsd:attribute name="connection" type="xsd:string"/>
    <xsd:attribute name="create" type="xsd:dateTime"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *id* attribute SHOULD represent the identifier of the transaction. Several transaction elements that belong to a transaction process SHOLD have same id value. For example, transaction elements in the same messaging model have the same id value. Re-sending depending on errors SHOULD have the same transaction id as the previous one. Every transaction process SHOULD have a unique id in the scope of the sender or the requester.
- *type* attribute SHOULD represent transaction control type. “Start” SHOLD represent to start transaction, while “Commit” SHOULD represent commitment and finalize the transaction. If the value is “Cancel”, then it SHOULD represent that the transaction is canceled and the process stops.
- *confirm* attribute SHOULD represent a confirmation request. The value of the attribute MSUT be either “Never”, “OnError”, or “Always”.
- *create* attribute SHOULD represent a date when the transaction is created.
- *description* attribute SHOULD represent any comments or descriptions.

Elements under the transaction element SHOULD follow the sentences:

- *Document* element SHOULD represent domain document to process in the responder.

6.3 Document element

Domain document is information unit to perform actions by application programs. Domain document is represented by document element. The specific list of domain documents which are necessary for production planning and scheduling can be described by application profile [PPS03].

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="Document">
  <xsd:complexType>
    <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="name" type="xsd:string" use="required"/>
    <xsd:attribute name="ref" type="xsd:string"/>
    <xsd:attribute name="action" type="xsd:string"/>
    <xsd:attribute name="option" type="xsd:string"/>
    <xsd:attribute name="event" type="xsd:string"/>
    <xsd:attribute name="namespace" type="xsd:string"/>
    <xsd:attribute name="create" type="xsd:dateTime"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *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.
- *name* attribute SHOULD represent name of domain document. The name SHOULD be selected from the list in the application profile.
- *ref* attribute SHOULD represent the identifier of a primary message document or other document that is in the same transaction element, when the transaction element has more than one 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”.
- *option* attribute SHOULD represent any optional information that may be interpreted by the receiver of the message.
- *event* SHOULD represent the identifier of event. When the document requests synchronization message, this value show the name of event the responder show in the profile. Notify document of the event also has the event name in this attribute.
- *namespace* attribute SHOULD represent namespace of the name of this document. When the implementation profile of the sender application supports more than one namespace, this attribute is required to identify the corresponding profile.
- *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 document. This element is defined in [PPS01].
- *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 of the document independently defined from the domain objects.

- *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.

Action type that the document element has in its action attribute determines the structure of the element available to specify. The table below shows the combination matrix. Each column shows different document action type, while the row shows available elements in the document 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 document 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	

6.4 Error element

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

The *Document* element SHOULD have one or more *Error* elements if the document is sent as error information. The *Document* element SHOULD NOT have an *Error* element if the document 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 document 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 document”.
  - “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 document is error notification.
  - “Warning” represents that the document is warning.
- *description* attribute SHOULD represent any description of the error explanations.

6.5 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 under 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>
```

6.6 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 property values.

If there is 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 [PCRE].
- *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

6.7 Selection element

Selection element SHOUD represent information for appropriate properties to be selected in the all domain properties in the domain object. *Selection* elements are used in Get documents and Change documents.

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

When the target property of selection is multiple, then the parent Get document or Change document is required for each attribute object that the multiple property is defined.

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" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="multiple" type="xsd:boolean"/>
    <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 “true”. Change or Get document 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 document in PULL model, the corresponding information of this property is addressed in the body of the response document. More than one *Property* elements which represent multiple property SHOULD NOT be described in the same *Selection* element.

- *type* attribute SHOULD represent the type of action after selecting the target properties. The available values are defined depending on the type of document.
  - “Insert” for Change document SHOULD represent that the property value is inserted, this is default value. This value SHOULD NOT be described in Get document.
  - “Update” for Change document SHOULD represent that the property value is updated. This value SHOULD NOT be described in Get document.
  - “Delete” for Change document SHOULD represent that the property value is deleted. This value SHOULD NOT be described in Get document.
  - “None” for Get document SHOULD represent that the target is specified by *Property* element. This is default value. This value SHOULD NOT be described in Change document.
  - “Typical” for Get document SHOULD represent that the target property is typical set. This value SHOULD NOT be described in Change document.
  - “All” for Get document SHOULD represent that the target property is all properties in the object. This value SHOULD NOT be described in Change document.
- *multiple* attribute for Get document SHOULD show whether the selected property is regarded as multiple or single one. If application profile or implementation profile shows that the property is single, then the selected property is regarded as single. No description of this attribute SHOULD represent single property.
- *count* attribute for Get document SHOULD represent the maximum number of properties selected by the *Property* element for the domain object. This value SHOULD NOT be described in Change document. This value SHOULD NOT be described for single property suggested by *multiple* attribute.
- *offset* attribute for Get document SHOULD represent the number of skipping the properties selected by the *Property* element for the domain object. This value SHOULD NOT be described in Change document. This value SHOULD NOT be described for single property suggested by *multiple* attribute.

6.8 Header element

Header element is used for representing header information in Show and Notify documents. The header information is described for any data depending on the document from an entire perspective. In Get document, *Header* element MAY be used to make brief inquiry of domain object that is not in the target of domain document. The *Header* element SHOULD be described in document elements.

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 document.

- *id* attribute SHOULD represent ID of the target object that is shown in the header by describing its property in the “Property” element.
- *class* attribute SHOULD represent the target domain object that the header shows the information in its *Property* elements. If there is no class attribute, then it represents that the target domain object is those that the domain document refers to as default.
- *title* attribute SHOULD represent a title of the document.
- *count* attribute SHOULD represent the number of domain objects in the document. When this attribute is used in Notify document and Show document, the value equals to the number of object in the body of the document. In Get document, the value represents the maximum number of objects the receiver is expecting in the Show document.
- *offset* attribute SHOULD represent the offset number of data list. When the objects in the document are not all of the existing objects in the sender, the offset value shows the relative position of the first object on the document body in the whole objects. This attribute can be used in Get document as a request to offset the response data. In Notify and Show document, this value shows the offset number of the body.

6.9 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 objects. When *Selection* element has a *Property* element, it shows the target property of changing or getting documents. 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" maxOccurs="unbounded"/>
      <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>
    </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*, and *Time* elements SHOULD represent a value of the property. These elements is defined in [PPS01]. When the property is described in *Condition* elements, constraint of property value MAY be described, where the value attribute in *Qty*, *Char*, and *Time* element shows the

value of constraints, and condition attribute in *Qty*, *Char*, and *Time* element shows constraint type. Multiple constraints under one property SHOULD be regarded conjunctive.

- *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 in Selection element and Header element. When this attribute is described, then the value described in Qty, Char and Time SHOULD be ignored. When the data type of this attribute is Qty or Time, then the value needs to be parsed to the corresponding data type.
- *sort* attribute SHOULD represent that the objects in the body of this document are expected to be sorted by ascending or descending order. For Get document, this attribute SHOULD be used in under *Selection* element. For Show document and Notify document, this attribute SHOULD be specified in *Header* element. If more than one *Property* element that has sort attribute are described in *Get* document, these sort requests SHOULD be applied in the priority rule that the 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 document. For Get document, this attribute SHOULD be used in *Selection* element. For Show document and Notify document, this attribute SHOULD be described 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 text string that can be shown in the header line for each primitive for explanation. This attribute is used only under the *Header* element.

A. Implementation level (Normative)

Since this specification provides the highest level functionality of application programs of 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. Regarding such situation, this specification additionally defines implementation levels for each function.

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 can know available combination of messaging without making a configuration tests.

The following table prescribes the implementation levels.

Table 4 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. In this specification, "Level 2 Function" denote that the section or subsection is not necessary for the application program that declares level 1 for the messaging model.

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