Production Planning and Scheduling (PPS) Version 1.0

Committee Specification Draft 01

02 June 2011

Specification URIs:

This version:

http://docs.oasis-open.org/pps/pps/v1.0/csd01/pps-v1.0-csd01.pdf (Authoritative)
http://docs.oasis-open.org/pps/pps/v1.0/csd01/pps-v1.0-csd01.html
http://docs.oasis-open.org/pps/pps/v1.0/csd01/pps-v1.0-csd01.doc

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/pps/pps/v1.0/pps-v1.0.pdf (Authoritative)
http://docs.oasis-open.org/pps/pps/v1.0/pps-v1.0.html
http://docs.oasis-open.org/pps/pps/v1.0/pps-v1.0.doc

Technical Committee:

OASIS Production Planning and Scheduling TC

Chair:

Yasuyuki Nishioka, PSLX Forum / Hosei University

Editors:

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

Related work:

This specification replaces or supersedes:

- PPS (Production Planning and Scheduling) Part 1: Core Elements, Version 1.0
- PPS (Production Planning and Scheduling) Part 2: Transaction Messages, Version 1.0
- PPS (Production Planning and Scheduling) Part 3: Profile Specifications, Version 1.0

This specification is related to:

- XML schema: pps/v1.0/csd01/xsd/

Declared XML namespace:

http://docs.oasis-open.org/ns/pps/2011

Abstract:

OASIS Production Planning and Scheduling (PPS) specification deals with problems of decision-making in all manufacturing companies who want to have a sophisticated information system for production planning and scheduling. PPS specification provides XML schema and communication protocols for information exchange among manufacturing application programs in the web-services environment. The Core Elements section focuses on information model of core elements which can be used as ontology in the production planning and scheduling domain. Since the elements have been designed without particular contexts in planning and scheduling, they can be used in any specific type of messages as a building block depending on the context of application programs. The Transaction Messages section focuses on transaction messages that represent

domain information sent or received 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. Finally, the Profile Specifications section focuses on profiles of application programs that may exchange the messages. Application profile and implementation profile are defined. Implementation profile shows capability of application programs in terms of services for message exchange, selecting from all exchange items defined in the application profile. The profile can be used for definition of a minimum level of implementation of application programs which are involved in a community of data exchange.

Status:

This document was last revised or approved by the OASIS Production Planning and Scheduling TC on the above date. The level of approval is also listed above. Check the "Latest 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).

Citation format:

When referencing this specification the following citation format should be used:

[PPS]

Production Planning and Scheduling (PPS) Version 1.0. 02 June 2011. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/pps/pps/v1.0/csd01/pps-v1.0-csd01.html.

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

Table of Contents

1 Introduction

This specification focuses on production planning and scheduling for all kinds of products and services provided by manufacturing enterprises. Production scheduling applications dealt in this specification can be divided into scheduling in the whole enterprise including some areas and sites, and detailed scheduling within an individual area and work-centers.

The scope of this specification, however, doesn't include optimization logic for solution, special knowledge of individual enterprises, concrete solution methods for production planning and scheduling, and planning problems for the total supply chain.

Section 2 of this specification prescribes how to describe contents of the XML messages which are used for exchanging the information on Production Planning and Scheduling by some application software programs.

If information defined with PPS is exchanged between production planning and scheduling applications, the enterprise can develop systems easily at a low cost and make them more competitive for the whole enterprise. In order to do this, the systems have to have high extendability as well.

Section 3 of this specification provides structure and rules of XML transaction elements for messaging between two application programs. Main parts of XML representations of the messages consist of XML core elements defined in Section 2. Those specifications define 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. Section 3 prescribes communication protocols by categorizing such various transactions into simple models. The specification doesn't focus on the underlying communication protocols, such as HTTP, SMTP and FTP.

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 Section 2.

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.

Section 4 of this specification prescribes definition of application profile and implementation profile. Implementation profile shows capability of information exchange with other application programs using PPS transaction messages. In order to define an implementation profile for each application program, this document also defines and prescribes application profile specification that should be consistent with all implementation profiles. An application profile allows each individual program to describe their capability.

Application profile shows a set of domain documents, domain objects and domain properties, which may be used in a message of production planning and scheduling application programs. Implementation profile shows domain documents, domain objects and domain properties that the application program can deal with correctly. The implementation profile also shows an implementation level of the application program. By collecting implementation profiles, a system integrator can arrange particular messaging in accordance with application specific scenarios.

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.

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

[PATH] XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath

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 Terms and definitions

Plan

Unit for intensive information of related orders corresponding to a specific period on a discrete time scale, or calculated information based on the schedule under the related orders. This can represent actual results when the related events have been occurred.

Order

Unit of requirement describing concrete item, resource or operation in a specific place at a specific time. This can also represent the results to the requirement.

Party

Customer who is a sender of an order and has a demand to make a decision, or supplier who is a receiver in case that a decision-maker sends the demand that can't be handled inside.

Item

Object to be produced or consumed by production activities. The quantity or the quality of item is changed during the production activity. Examples include product, parts, module, unit, work in process and materials.

Resource

Object that can provide essential function for production activities. The capacity of function is used during production activity, and is available again after finishing the production. Examples include equipment, machine, device, labor and tool.

Process

Segment of production activities indicating a certain production line or method. This takes duration from start time to end time, and gives added value to the producing item. One process may have two or more than two processes detailed in the lower levels.

Lot

Instance of a specific volume of item that exists in a specific place at a specific time. Generally the specific time corresponds to start or end of an operation, and the specific volume is equal to the quantity of item produced or consumed by the operation.

Task

Unit of necessity to execute a specific operation at a specific time, indicating the volume of used capability provided by the applicable resource. This can represent both capacity value provided

by resource at a specific time point, and aggregated total value of capacity provided by resource during specific duration.

Operation

Actual processing element to be executed by a specific task, and to produce or consume a specific lot. It is a concrete instance of particular processes in production activities.

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 this specification. 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 Core Elements

2.1 Primitive Elements

2.1.1 Structure of primitive elements

Primitive elements are the minimum series of element that corresponds to the most basic domain objects. The type of this element MUST be represented with the following XML schema.

```
<xsd:complexType name="PrimitiveType">
  <xsd:sequence>
    <xsd:element ref="Compose" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Produce" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Consume" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Assign" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Relation" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Location" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Capacity" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Progress" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Spec" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Start" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="End" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Event" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Price" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Cost" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="key" type="xsd:long"/>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="parent" type="xsd:string"/>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="status" type="xsd:string"/>
  <xsd:attribute name="party" type="xsd:string"/>
  <xsd:attribute name="plan" type="xsd:string"/>
  <xsd:attribute name="order" type="xsd:string"/>
  <xsd:attribute name="item" type="xsd:string"/>
  <xsd:attribute name="resource" type="xsd:string"/>
  <xsd:attribute name="process" type="xsd:string"/>
  <xsd:attribute name="lot" type="xsd:string"/>
  <xsd:attribute name="task" type="xsd:string"/>
  <xsd:attribute name="operation" type="xsd:string"/>
</xsd:complexType>
```

- *id* attribute SHOULD represent an identifier of the domain object.
- *key* attribute represents a key used in the local applications.
- *name* attribute represents the name of the domain object.
- *parent* attribute represents the identifier of the inherited object of the domain object.
- *type* attribute represents the modifier of the domain object.
- *status* attribute represents the status of the domain object.
- *party* attribute represents an identifier of the party associated with the domain object.
- *plan* attribute represents the identifier of the plan associated with the domain object.
- *order* attribute represents the identifier of the order associated with the domain object.

- *item* attribute represents the identifier of the item associated with the domain object.
- *resource* attribute represents the identifier of the resource associated with the domain object.
- *process* attribute represents the identifier of the process associated with the domain object.
- *lot* attribute represents the identifier of the lot associated with the domain object.
- *task* attribute represents the identifier of the task associated with the domain object.
- *operation* attribute represents the identifier of the operation associated with the domain object.

- *Compose* element represents the element corresponding to part of the domain object.
- *Produce* element represents the relation that the domain object produces.
- *Consume* element represents the relation that the domain object consumes.
- *Assign* element represents the relation that the domain object uses.
- *Relation* element represents the relation to other primitive elements.
- *Location* element represents the location where the domain object exists.
- *Capacity* element represents the capacity status of the domain object.
- *Progress* element represents the progress of the domain object.
- *Spec* element represents the specification of the domain object.
- *Start* element represents the start event of the domain object.
- *End* element represents the completion event of the domain object.
- *Event* element represents the optional event under the domain object.
- *Price* element represents the price of the domain object.
- *Cost* element represents the cost of the domain object.
- *Priority* element represents the priority of the domain object.
- *Display* element represents how to display the domain object.
- *Description* element represents the description of the domain object.
- *Author* element represents the author of the domain object information.
- *Date* element represents the date of the domain object information.

2.1.2 List of primitive elements

This specification defines nine primitive elements: *Party*, *Plan*, *Order*, *Item*, *Resource*, *Process*, *Lot*, *Task*, and *Operation*. The type of those elements MUST be represented with the following XML schema.

```
<xsd:element name="Party" type="PrimitiveType"/>
<xsd:element name="Plan" type="PrimitiveType"/>
<xsd:element name="Order" type="PrimitiveType"/>
<xsd:element name="Item" type="PrimitiveType"/>
<xsd:element name="Resource" type="PrimitiveType"/>
<xsd:element name="Process" type="PrimitiveType"/>
<xsd:element name="Lot" type="PrimitiveType"/>
<xsd:element name="Task" type="PrimitiveType"/>
<xsd:element name="Operation" type="PrimitiveType"/>
```

2.1.2.1 Party element

Party element represents a customer or a supplier. Customer is an object that requests some products or services to the enterprise. The requests are sent to a person who is in charge of production planning and

scheduling. Supplier is an object providing some products or services to the enterprise. Supplier receives orders from the enterprise, and provides corresponding items, resources or processes for the enterprise.

2.1.2.2 Plan element

Plan element represents a value planned for particular products or services. The value shows volume of the products or services required or resulted during certain period of time. Typical cases of planning period include day, week and month.

2.1.2.3 Order element

Order element represents an object of information produced to request some products or services. Order is source to create production orders that are finally dispatched to the plant floor. Orders can be divided into inventory order, capacity order and production order according to the type of request.

Example: Item “A” products are requested.

```
<Order id=”Z01” item=”A”>
  <Spec type=”quantity”><Qty value=”10”/></Spec>
</Order>
```

Example: Three labors in ”group B” are requested.

```
<Order id=”Z02” resource=”groupB”>
  <Spec type=”quantity”><Qty value=”3”/></Spec>
</Order>
```

Example: Switching operation is requested two times.

```
<Order id=”Z03” process=”change01”>
  <Spec type=”quantity”><Qty value=”2”/></Spec>
</Order>
```

Example: Order which consist of 10 of “A” and 5 of “B” is totally 3,000 yen.

```
<Order id=”Z00”>
  <Compose order=”Z01”/>
  <Compose order=”Z02”/>
  <Price value=”3000” unit=”yen”/>
</Order>
<Order id=”Z01” item=”A”>
  <Spec type=”quantity”><Qty value=”10”/></Spec>
</Order>
<Order id=”Z02” item=”B”>
  <Spec type=”quantity”><Qty value=”5”/></Spec>
</Order>
```

2.1.2.4 Item element

Item element represents a product, component, parts, work in process (WIP), raw material and other items. Item is produced by any processes, and after that, it is consumed by another processes.

2.1.2.5 Resource element

Resource element represents a resource, which is an object enabling production, transportation, storage, inspection and other various services. As resource can produce tasks to execute operations, it is assigned to an operation by considering its volume of capacity.

2.1.2.6 Process element

Process element represents a process that has a function to produce value. Process can be defined as a segment of activities in production process. It produces and consumes production items by being executed during certain period of time.

2.1.2.7 Lot element

Lot element represents a production lot. Production lot is an object corresponding to a concrete item that actually exists in a specific place at a specific date and time. Lot is produced by an operation and finally consumed by another operation or discarded.

2.1.2.8 Task element

Task element represents a task, which is an object showing the usage of a specific resource capability for a specific period of time. Schedule may request a certain volume of task for each resource assigned to execute the appropriate operations.

Example: Task corresponding to the volume that 3 labors work load is required for 2 days

```
<Task id=”T01”>
  <Capacity type=”human”><Qty value=”3”/></Capacity>
  <Capacity type=”duration”><Qty value=”2” unit=”day” /></Capacity>
</Task>
```

2.1.2.9 Operation element

Operation element represents a segment of activities that is actually dispatched to plant floor. Operation identifies an executable function at a specific place on a plant floor for a specific time. Operation is associated with a specific lot and task by executing those activities.

2.2 Relational Elements

2.2.1 Structure of relational elements

Relational elements represent any relations between domain objects. A relational element can have properties. The type of this element MUST be represented with the following XML schema.

```
  <xsd:complexType name="RelationalType">
    <xsd:sequence>
      <xsd:element ref="Location" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Capacity" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Progress" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Spec" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Start" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="End" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Event" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Price" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Cost" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
      <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:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="key" type="xsd:long"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="status" type="xsd:string"/>
    <xsd:attribute name="apply" type="xsd:string"/>
    <xsd:attribute name="party" type="xsd:string"/>
    <xsd:attribute name="plan" type="xsd:string"/>
    <xsd:attribute name="order" type="xsd:string"/>
    <xsd:attribute name="item" type="xsd:string"/>
```

```
    <xsd:attribute name="resource" type="xsd:string"/>
    <xsd:attribute name="process" type="xsd:string"/>
    <xsd:attribute name="lot" type="xsd:string"/>
    <xsd:attribute name="task" type="xsd:string"/>
    <xsd:attribute name="operation" type="xsd:string"/>
  </xsd:complexType>
```

- *id* attribute SHOULD represent an identifier of the relation.
- *key* attribute represents a key used in the local applications.
- *name* attribute represents the name of the relation.
- *type* attribute represents the modifier of the relation.
- *status* attribute represents the status of the relation.
- *apply* attribute represents application type of the relation. This element is a disjunctive (OR) content under the parent object, if the attribute value is "*disjunctive* ".
- *party* attribute represents an identifier of the party associated with the relation.
- *plan* attribute represents the identifier of the plan associated with the relation.
- *order* attribute represents the identifier of the order associated with the relation.
- *item* attribute represents the identifier of the item associated with the relation.
- *resource* attribute represents the identifier of the resource associated with the relation.
- *process* attribute represents the identifier of the process associated with the relation.
- *lot* attribute represents the identifier of the lot associated with the relation.
- *task* attribute represents the identifier of the task associated with the relation.
- *operation* attribute represents the identifier of the operation associated with the relation.

- *Location* element represents the location associated with the relation.
- *Capacity* element represents the capacity status of the relation.
- *Progress* element represents the progress of the relation.
- *Spec* element represents the specification of the relation.
- *Start* element represents the start event of the relation.
- *End* element represents the completion event of the relation.
- *Event* element represents the optional event under the relation.
- *Price* element represents the price of the relation.
- *Cost* element represents the cost of the relation.
- *Priority* element represents the priority of the relation.
- *Display* element represents how to display the relation.
- *Description* element represents the description of the relation.
- *Author* element represents the author of the relation information.
- *Date* element represents the date of the relation information.
- *Qty* element represents the quantity of the relation.
- *Char* element represents the qualitative value of the relation.
- *Time* element represents the time of the relation.

2.2.2 List of relational elements

This part of specifications defines five relational elements: *Compose*, *Produce*, *Consume*, *Assign*, and *Relation*. Relational element defines relationship between the parent element and those that characterize the element. The type of this element MUST be represented with the following XML schema.

```
  <xsd:element name="Compose" type="RelationalType"/>
  <xsd:element name="Produce" type="RelationalType"/>
  <xsd:element name="Consume" type="RelationalType"/>
  <xsd:element name="Assign" type="RelationalType"/>
  <xsd:element name="Relation" type="RelationalType"/>
```

2.2.2.1 Compose element

Compose element defines a hierarchical relation between the parent element and another same primitive element that addresses one level upper or lower than the target element. This element can represent that the object referred to in this element composes or be composed by the parent element.

Example: Product “A” family includes product “A1” and product “A2”.

```
<Item id=”A”>
  <Compose type=”child” item=”A1”/>
  <Compose type=”child” item=”A2”/>
</Item>
```

Example: Product “B” is assembled with 2 of parts “C1” and 3 of parts “C2”.

```
<Item id=”B”>
  <Compose type=”child” item=”C1”><Qty value=”2”/></Compose>
  <Compose type=”child” item=”C2”><Qty value=”3”/></Compose>
</Item>
```

Example: 2 of parts “C1” are used for product “B1”, and 5 of parts “C1” are used for product “B2”.

```
<Item id=”C1”>
  <Compose type=”parent” item=”B1”><Qty value=”2”/></Compose>
  <Compose type=”parent” item=”B2”><Qty value=”5”/></Compose>
</Item>
```

2.2.2.2 Produce element

Produce element defines a relation between processes and items, or a relation between operations and lots. This element can show the quantity of the item or lot produced by the process or operation respectively, or how many items or lots are produced by the process or the operation respectively.

2.2.2.3 Consume element

Consume element defines a relation between processes and items, or a relation between operations and lots. This element can show the quantity of the item or lot consumed by the process or operation respectively, or how many items or lots are consumed by the process or operation respectively.

2.2.2.4 Assign element

Assign element defines a relation between processes and resources, or a relation between operations and tasks. This element can show the volume of capacity provided by the resource or task assigned for the process or operation respectively, or how many resources or tasks are used.

2.2.2.5 Relation element

Relation element can show that the parent element has a specific relation to other primitive elements. This element can additionally define relational classes between primitive elements. Examples include precedence relations and pegging relations.

2.3 Specific Elements

2.3.1 Structure of specific element

Specific elements are defined to represent any properties of the primitive element. This element MAY be described more than once on the same parent element if the value is historical. Those multiple properties have time stamp. The type of this element MUST be represented with the following XML schema.

```
<xsd:complexType name="SpecificType">
  <xsd:sequence>
    <xsd:element ref="Start" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="End" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Event" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Price" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Cost" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
    <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:sequence>
  <xsd:attribute name="id" type="xsd:string"/>
  <xsd:attribute name="key" type="xsd:long"/>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="status" type="xsd:string"/>
  <xsd:attribute name="apply" type="xsd:string"/>
</xsd:complexType>
```

- *id* attribute SHOULD represent an identifier of the property.
- *key* attribute represents a key used in the local applications.
- *name* attribute represents the name of the property.
- *type* attribute represents the modifier of the property.
- *status* attribute represents the status of the property.
- *apply* attribute represents application type of the property. The value of the element is relative, if the value is "*relative* ".

- *Start* element represents the start event of the property.
- *End* element represents the completion event of the property.
- *Event* element represents the optional event under the property.
- *Price* element represents the price of the property.
- Cost element represents the cost of the property.
- *Priority* element represents the priority of the property.
- *Display* element represents how to display the property.
- *Description* element represents the description of the property.

- *Author* element represents the author of the property information.
- *Date* element represents the date of the property information.
- *Qty* element represents the quantity of the property.
- *Char* element represents the qualitative value of the property.
- *Time* element represents the time of the property.

2.3.2 List of specific elements

For specific elements, this part of specifications has four elements: *Location*, *Capacity*, *Progress*, and *Spec*. The type of this element MUST be represented with the following XML schema.

```
  <xsd:element name="Location" type="SpecificType"/>
  <xsd:element name="Capacity" type="SpecificType"/>
  <xsd:element name="Progress" type="SpecificType"/>
  <xsd:element name="Spec" type="SpecificType"/>
```

2.3.2.1 Location element

Location element represents a location. When the expression of location has structure, multiple values can be set by describing different names of the data. Change of the location depending on time can also be represented by multiple values.

Example: Customer's address

```
<Party id="ABC Inc.">
  <Location type="address"><Char value="123 ABC street"/></Location>
  <Location type="city"><Char value="Cambridge"/></Location>
  <Location type="state"><Char value="MA"/></Location>
  <Location type="code"><Char value="02139"/></Location>
  <Location type="country"><Char value="USA"/></Location>
</Party>
```

2.3.2.2 Capacity element

Capacity element represents volume of capability provided by resources, items or processes. In the case of resource capability, it may show available amount of corresponding tasks. In the case of Items, it shows the available amount of Lots. And for Processes, it shows maximum ratio of production. All of this information is represented in a time horizon.

Example: Inventory level of "material01"

```
<Item id="material01">
  <Capacity><Qty value="150"/></Capacity>
</Item>
```

Example: Temporal change of the material

```
<Item id="material01">
  <Capacity><Qty value="150"><Time value="2005-04-10T00:00:00/></Capacity>
  <Capacity><Qty value="200"><Time value="2005-04-17T00:00:00/></Capacity>
</Item>
```

Example: Material location information: Stock of "material01" is 150 located at "storage01"

```
<Item id="material01">
```

```
  <Location value=”storage01”/>
  <Capacity><Qty value=”150”/></Capacity>
</Item>
```

2.3.2.3 Progress element

Progress element represents progress of order and operation, or status of lot and task. This element shows the latest data, status or progress at a specific time point. This element MAY represent a change of time-dependent values.

2.3.2.4 Spec element

Spec element represents various specifications for primitive elements. The content can be represented with a pair of a spec name and a value. This element can also represent time-dependent change of the value. The value of the specification is represented with one data type of a numerical value, characters and date time.

2.4 Eventual Elements

2.4.1 Structure of eventual element

Eventual elements represent any properties that occur at one time point. Any type of events can be specified by using this element. The type of this element MUST be represented with the following XML schema.

```
  <xsd:complexType name="EventualType">
    <xsd:sequence>
      <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
      <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:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="key" type="xsd:long"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="status" type="xsd:string"/>
    <xsd:attribute name="apply" type="xsd:string"/>
    <xsd:attribute name="condition" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string"/>
  </xsd:complexType>
```

- *id* attribute SHOULD represent an identifier of the property.
- *key* attribute represents a key used in the local applications.
- *name* attribute represents the name of the property.
- *type* attribute represents the modifier of the property.
- *status* attribute represents the status of the property.
- *apply* attribute represents application type of the property. The value of this element is exclusive, if the value is “exclusive”.
- *condition* attribute represents the condition of the property.
- *value* attribute represents the qualitative value of the property.

- *Priority* element represents the priority of the property.
- *Display* element represents how to display the property.
- *Description* element represents the description of the property.
- *Author* element represents the author of the property information.
- *Date* element represents the date of the property information.
- *Qty* element represents the quantity of the property.
- *Char* element represents the qualitative value of the property.
- *Time* element represents the time of the property.

2.4.2 List of eventual elements

This part of specifications defines three eventual elements: *Start*, *End*, and *Event*. The *Start* and *End* are special cases of *Event* element. The type of this element MUST be represented with the following XML schema.

```
<xsd:element name="Start" type="EventualType"/>
<xsd:element name="End" type="EventualType"/>
<xsd:element name="Event" type="EventualType"/>
```

2.4.2.1 Start element

Start element represents a start event of orders, processes or operations. In case of order, this element represents an event at the earliest start time of corresponding operations.

2.4.2.2 End element

End element represents an end event of orders, processes or operations. In case of order, this element represents an event at the latest end time of corresponding operations.

2.4.2.3 Event element

Event element represents an event associated with a customer, supplier, item, resource, process or operation. Event brings any action or any status change at a specific time point. In general, the status value of item or resource changes discontinuously before the event.

2.5 Accounting Elements

2.5.1 Structure of Accounting element

Accounting element represents any accounting information such as profit revenue and cost spending. Price and cost associated with goods and services are the target of the elements. The type of this element MUST be represented with the following XML schema.

```
<xsd:complexType name="AccountingType">
  <xsd:sequence>
    <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
```

```
      <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
      <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:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="key" type="xsd:long"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="status" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string"/>
    <xsd:attribute name="condition" type="xsd:string"/>
    <xsd:attribute name="apply" type="xsd:string"/>
  </xsd:complexType>
```

- *id* attribute SHOULD represent an identifier of the property.
- *key* attribute represents a key used in the local applications.
- *name* attribute represents the name of the property.
- *type* attribute represents the modifier of the property.
- *status* attribute represents the status of the property.
- *apply* attribute represents application type of the property. The value of this element is exclusive, if the value is “exclusive”.
- *condition* attribute represents the condition of the property.
- *value* attribute represents the qualitative value of the property.

- *Priority* element represents the priority of the property.
- *Display* element represents how to display the property.
- *Description* element represents the description of the property.
- *Author* element represents the author of the property information.
- *Date* element represents the date of the property information.
- *Qty* element represents the quantitative value of the property.
- *Char* element represents the qualitative value of the property.
- *Time* element represents the temporal value of the property.

2.5.2 List of accounting elements

For accounting elements, *Price* element and *Cost* element are defined in this specification. The type of this element MUST be represented with the following XML schema.

```
  <xsd:element name="Price" type="AccountingType"/>
  <xsd:element name="Cost" type="AccountingType"/>
```

2.5.2.1 Price element

Price element represents a price. This element can be used to represent price information of primitive element and some properties.

2.5.2.2 Cost element

Cost element represents a cost. This element can be used to represent cost information of primitive element and some properties.

2.6 Administrative Elements

2.6.1 Structure of Administrative Elements

Administrative elements represent any administrative information, which is not the main body of the problem domain but the information how to deal with the domain information. The type of this element MUST be represented with the following XML schema.

```
<xsd:complexType name="AdministrativeType">
  <xsd:sequence>
    <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:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="status" type="xsd:string"/>
  <xsd:attribute name="apply" type="xsd:string"/>
  <xsd:attribute name="condition" type="xsd:string"/>
  <xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>
```

- *name* attribute represents the name of the property.
- *type* attribute represents the modifier of the property.
- *status* attribute represents the status of the property.
- *apply* attribute represents application type of the property. The value of this element is exclusive, if the value is “exclusive”.
- *condition* attribute represents the condition of the property.
- *value* attribute represents the qualitative value of the property.

- *Qty* element represents the quantitative value of the property.
- *Char* element represents the qualitative value of the property.
- *Time* element represents the temporal value of the property.

2.6.2 List of Administrative Elements

For administrative elements, *Priority*, *Display*, *Description*, *Author* and *Date* elements are defined in this specification. The type of this element MUST be represented with the following XML schema.

```
<xsd:element name="Priority" type="AdministrativeType"/>
<xsd:element name="Display" type="AdministrativeType"/>
<xsd:element name="Description" type="AdministrativeType"/>
<xsd:element name="Author" type="AdministrativeType"/>
<xsd:element name="Date" type="AdministrativeType"/>
```

2.6.2.1 Priority element

Priority element represents the priority of the primitive element or the parent element. This information is used to make a decision for planning or scheduling.

2.6.2.2 Display element

Display element is an element to set how to display the parent element. This element can specify colors or display locations on the screen.

2.6.2.3 Description element

Description element is an element to set an optional comment of the parent element. The comment data type is a character string.

2.6.2.4 Author element

Author element represents the author and its related information such as the authoring date. This information is not about the target domain model, but information processing model.

2.6.2.5 Date element

Date element is an element that shows the creation date, expire date, revising date, and so forth. This information is for administrative use of the domain model.

2.7 Data Elements

2.7.1 Qty element

Qty element SHOULD represent quantitative information. This element can be used to represent the quantitative numerical data by decimal type data format. Unit of the value can be set in this element, and representation of fraction is available. The type of this element MUST be represented with the following XML schema.

```
<xsd:element name="Qty">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="status" type="xsd:string"/>
    <xsd:attribute name="apply" type="xsd:string"/>
    <xsd:attribute name="condition" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:decimal"/>
    <xsd:attribute name="count" type="xsd:long"/>
    <xsd:attribute name="unit" type="xsd:string"/>
    <xsd:attribute name="base" type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```

- *name* attribute represents the name of the data.
- *type* attribute represents the modifier of the data.
- *status* attribute represents the status of the data.
- *apply* attribute represents application type of the data. The value of this element is exclusive, if the value is “exclusive”.
- *condition* attribute represents the condition of the data.
- *value* attribute represents the content corresponding to the qty element.
- *count* attribute represents the countable value of the data.

- *unit* attribute represents the type of unit of the data.
- *base* attribute represents the base data of the data. The value of the “value” attribute is divided with this value.

Example: 1/3 meters

```
<Qty value=”1”  unit=”m” base=”3”/>
```

Example: 3 weeks (discrete time scale)

```
<Qty count=”3” unit=”week” />
```

2.7.2 Char element

Char element SHOULD represent character data. This element can be used to represent a qualitative value of specification or a value of location. The type of this element MUST be represented with the following XML schema.

```
  <xsd:element name="Char">
    <xsd:complexType>
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:attribute name="type" type="xsd:string"/>
      <xsd:attribute name="status" type="xsd:string"/>
      <xsd:attribute name="apply" type="xsd:string"/>
      <xsd:attribute name="condition" type="xsd:string"/>
      <xsd:attribute name="value" type="xsd:string"/>
      <xsd:attribute name="count" type="xsd:long"/>
      <xsd:attribute name="unit" type="xsd:string"/>
      <xsd:attribute name="base" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
```

- *name* attribute represents the name of the data.
- *type* attribute represents the modifier of the data.
- *status* attribute represents the status of the data.
- *apply* attribute represents application type of the data. The value of this element is exclusive, if the value is “exclusive”.
- *condition* attribute represents the condition of the data.
- *value* attribute represents the content corresponding to the data.
- *count* attribute represents the countable value of the data.
- *unit* attribute represents the type of unit of the data.
- *base* attribute represents the base data of the data. The value of the “value” attribute is divided with this value.

2.7.3 Time element

Time element SHOULD represent a specific time. Time is represented by a continuous time scale, or a specific discrete time scale. The type of this element MUST be represented with the following XML schema.

```
  <xsd:element name="Time">
```

```
    <xsd:complexType>
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:attribute name="type" type="xsd:string"/>
      <xsd:attribute name="status" type="xsd:string"/>
      <xsd:attribute name="apply" type="xsd:string"/>
      <xsd:attribute name="condition" type="xsd:string"/>
      <xsd:attribute name="value" type="xsd:dateTime"/>
      <xsd:attribute name="count" type="xsd:long"/>
      <xsd:attribute name="unit" type="xsd:string"/>
      <xsd:attribute name="base" type="xsd:dateTime"/>
    </xsd:complexType>
  </xsd:element>
```

- *name* attribute represents the name of the data.
- *type* attribute represents the modifier of the data.
- *status* attribute represents the status of the data.
- *apply* attribute represents application type of the data. The value of this element is exclusive, if the value is “exclusive”.
- *condition* attribute represents the condition of the data.
- *value* attribute represents the content corresponding to the data.
- *count* attribute represents the countable value of the data.
- *unit* attribute represents the type of unit of the data.
- *base* attribute represents the base data of the data. The value of the “value” attribute is divided with this value.

Example: noon on May 13th, 2005

```
<Time value=”2005-05-13T12:00:00”/>
```

Example: 2 months later since the present month (May, 2005) (discrete time scale)

```
<Time count=”2” unit=”month” base=”2005-05-01T00:00:00”/>
```

3 Transaction Messages

3.1 Messaging model

3.1.1 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.

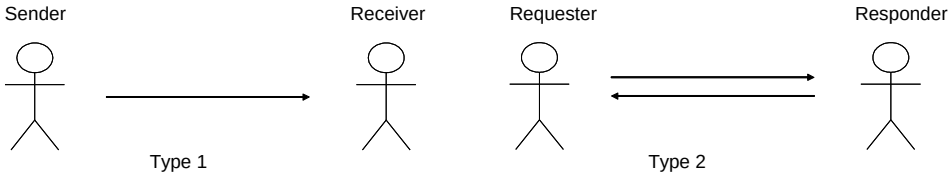


Figure 3.1 Basic unit of messaging

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 3.1. 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.

3.1.2 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 3.1 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 3.2 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.

3.1.3 Messaging models

3.1.3.1 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.

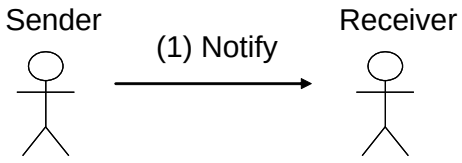


Figure3. 2 NOTIFY model

3.1.3.2 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.

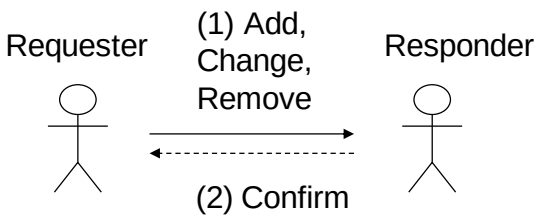


Figure 3.3 PULL model

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.

3.1.3.3 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.

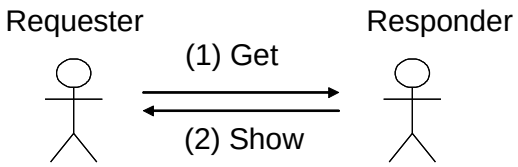


Figure 3.4 PULL model

3.1.3.4 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.

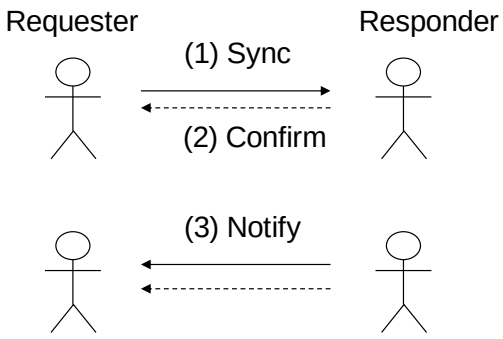


Figure 3.5 SYNC model

3.1.4 Procedures on responders

3.1.4.1 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.

3.1.4.2 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".

3.1.4.3 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 "Always" 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.

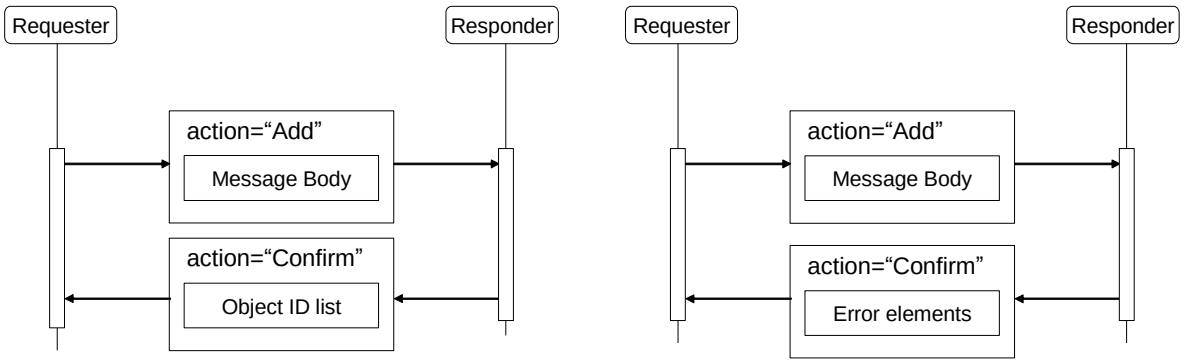
3.2 Add, Change and Remove (PUSH model)

3.2.1 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.



(a) Normal case (b) Error case

Figure 3.6 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>
```

3.2.2 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.

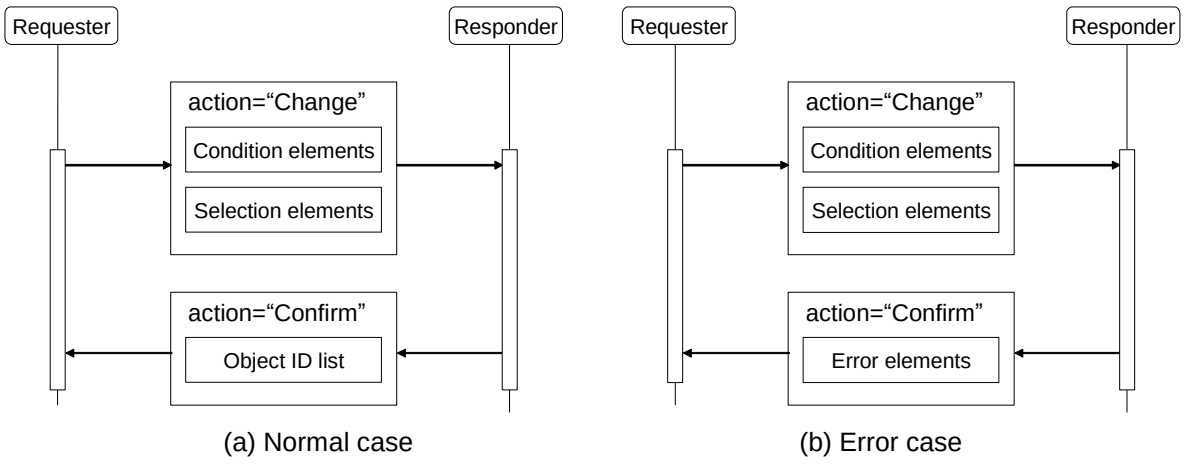


Figure 3.7 Change transactions

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.

3.2.2.1 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>
```

3.2.2.2 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.

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

Example: Revised status of the product data

```
<Document 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>
```

3.2.2.3 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” >
<Condition 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>
```

3.2.3 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.

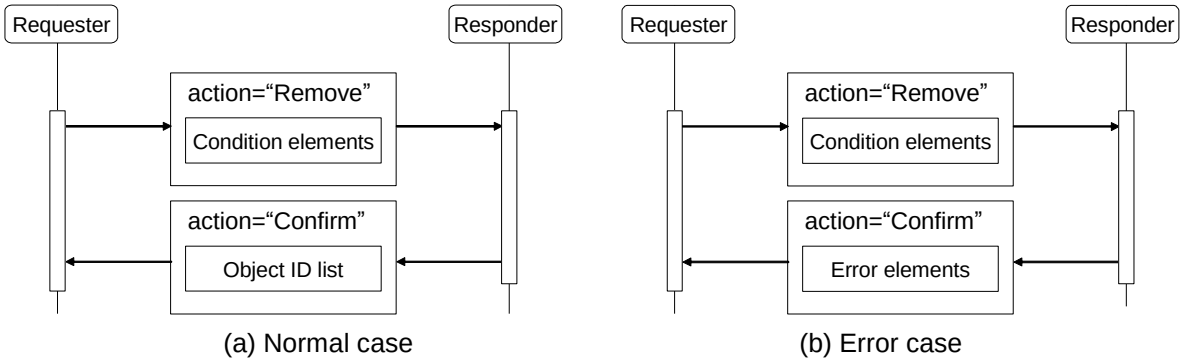


Figure 3.8 Remove transactions

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>
```

3.3 Notify and Sync (NOTIFY and SYNC model)

3.3.1 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.

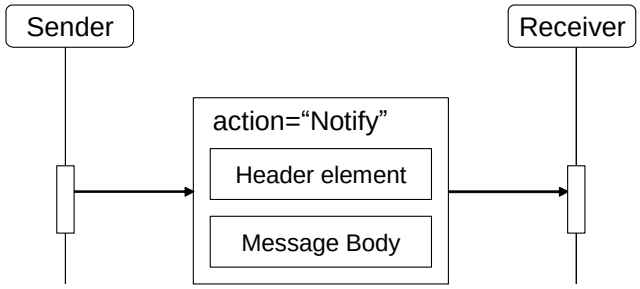


Figure 3.9 Notify transactions

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 Section 2, 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>
```

3.3.2 Synchronizing process

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.

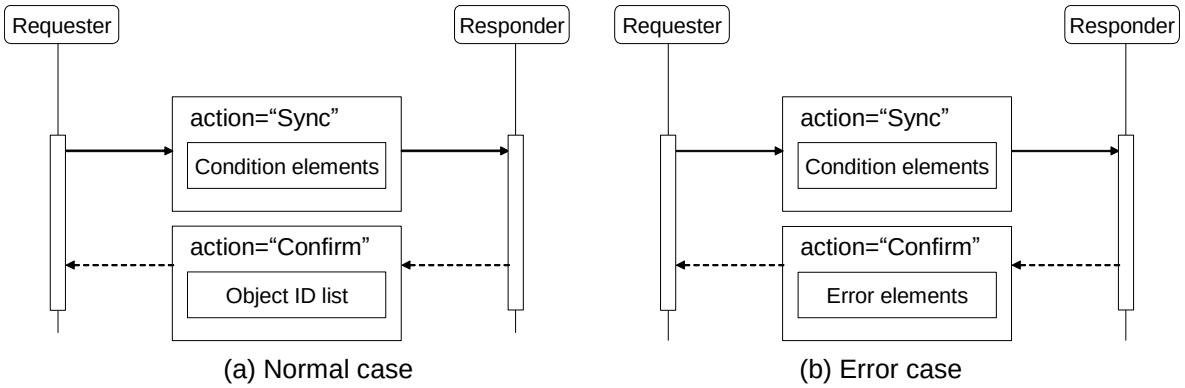


Figure 3.10 Sync transaction

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 Section 4 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.

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.

3.3.2.1 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 value=”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.

3.3.2.2 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 Section 4. 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.

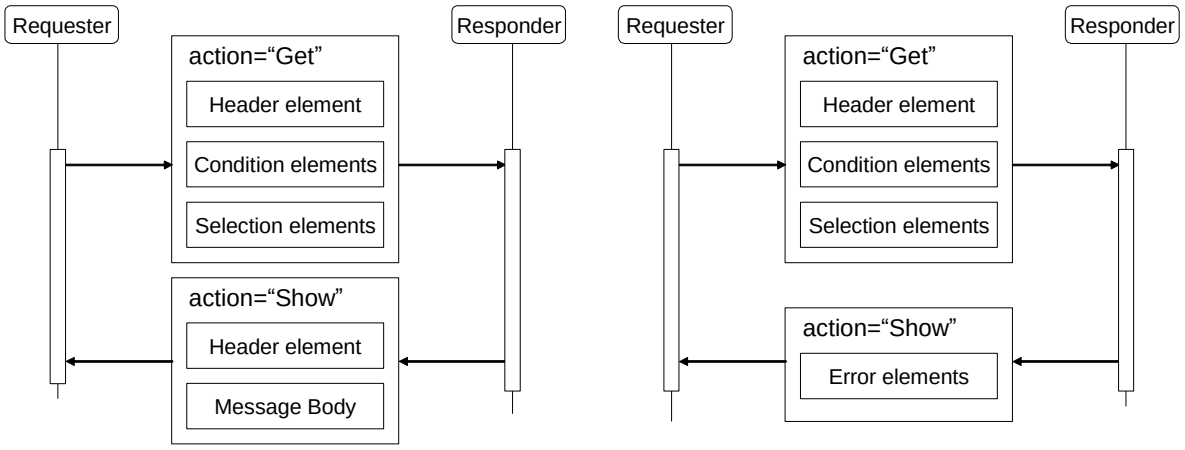
```
<Document id=”B-2” name=”ProductionOrder” action=”Notify” event=”E01” >
<Order id=”1204”>
  <Produce><Qty value=”200”/></Produce>
</Order>
</Document>
```

3.4 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 3.11 Get -Show transactions

3.4.1 Target domain objects

3.4.1.1 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>
```

3.4.1.2 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”/></Property>
</Condition>
<Selection type=”All”/>
</Document>
```

3.4.1.3 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>
```

3.4.1.4 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>
```

3.4.2 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.

3.4.2.1 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 defined in Section 4.

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>
```

3.4.2.2 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:".

3.4.2.3 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>
```

3.4.2.4 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

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

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.

3.4.3 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 3.12 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.

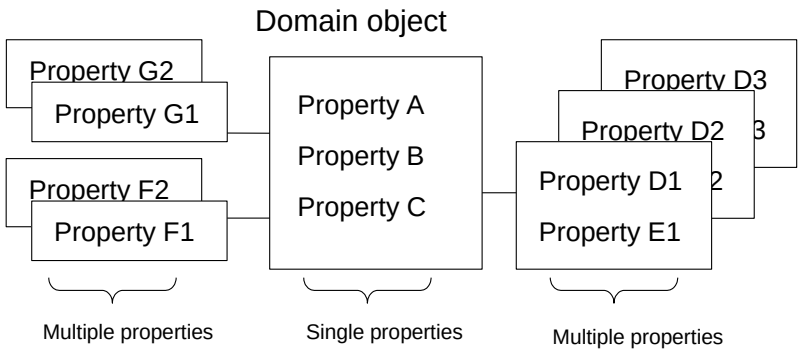


Figure 3.12: Single property and Multiple property

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.

3.4.4 Using Header element

3.4.4.1 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>
```

3.4.4.2 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>
```

3.4.5 Show document

3.4.5.1 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>
```

3.4.5.2 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” value=”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 value=”yen”/></Property>
  <Property type=”Condition” name=”pps:parent”><Char value=”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>
```

3.5 XML Elements

3.5.1 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 MUST be specified in the following XML schema.

```
 <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="security" 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 represents an identifier of the sender or requester of the message. This information is not for the low-level communication programs but for application programs.
- *security* attribute represents a security text data such as pass words for authorization of the sender.
- *create* attribute represents a date when the message is created.
- *description* attribute represents any comments or descriptions.

- Elements under this messageType element SHOULD follow the sentences:
- *ImplementProfile* element represents a request of implementation profile or answer of implementation profile defined in Section 4.
- *Transaction* element represents 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"/>
```

3.5.2 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 MUST be specified in the following XML schema.

```
<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 SHOULD 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 represents transaction control type. “Start” SHOULD 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 represents a confirmation request. The value of the attribute MUST be either “Never”, “OnError”, or “Always”.
- *create* attribute represents a date when the transaction is created.
- *description* attribute represents any comments or descriptions.

- Elements under the transaction element SHOULD follow the sentences:
- *Document* element represents domain document to process in the responder.

3.5.3 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 defined in Section 4.

This information MUST be specified in the following XML schema.

```
<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 represents name of domain document. The name SHOULD be selected from the list in the application profile.

- *ref* attribute represents 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 represents the type of the message, where the types correspond to verbs information for the message. Values of the attribute is either “Add”, “Change”, “Remove”, “Confirm”, “Notify”, “Sync”, “Get”, or “Show”.
- *option* attribute represents any optional information that may be interpreted by the receiver of the message.
- *event* represents 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 represents 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 represents a date when the transaction document is created.
- *description* attribute represents any comments or descriptions.

Elements under the transaction element SHOULD follow the sentences:

- *Error* element represents error information.
- *App* element represents any information for the application programs.
- *Spec* element represents any particular specification of the document. This element is defined in Section 2.
- *Condition* element represents any condition of selecting required domain objects.
- *Selection* element represents any condition of selecting required properties of a domain object.
- *Header* element represents information of the document independently defined from the domain objects.
- *Party*, *Plan*, *Order*, *Item*, *Resource*, *Process*, *Lot*, *Task*, or *Operation* element represent domain objects. Different type of them SHOULD NOT be specified at the same parent *Document* 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 3.3 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	

3.5.4 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 MUST 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 represents the document id that has the errors.
- *code* attribute represents unique identifier of the error categories. The error code MAY consist of three digits. If the first digit is 0, then the code MAY 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 represents the location of error texts.
- *status* attribute represents 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 represents any description of the error explanations.

3.5.5 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 MUST 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>
```

3.5.6 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 MUST 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 represents 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 represents the name of property that is used to apply wildcard value. The wildcard text is specified in the *value* attribute.
- *value* attribute represents the wildcard text for selecting the target domain objects. The text is interpreted by regular expression rules [PCRE].
- *version* attribute represents 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

3.5.7 Selection element

Selection element SHOULD 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 MUST 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 represents 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 represents 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 represents the type of action after selecting the target properties. The available values are defined depending on the type of document.
  - "Insert" for Change document represents that the property value is inserted, this is default value. This value is not described in Get document.
  - "Update" for Change document represents that the property value is updated. This value is not described in Get document.
  - "Delete" for Change document represents that the property value is deleted. This value is not described in Get document.
  - "None" for Get document can represent that the target is specified by *Property* element. This is default value. This value is not described in Change document.
  - "Typical" for Get document can represent that the target property is typical set. This value is not described in Change document.
  - "All" for Get document can represent that the target property is all properties in the object. This value is not described in Change document.
- *multiple* attribute for Get document shows 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 represents single property.

- *count* attribute for Get document represents the maximum number of properties selected by the *Property* element for the domain object. This value is not described in Change document. This value is not be described for single property suggested by *multiple* attribute.
- *offset* attribute for Get document represents the number of skipping the properties selected by the *Property* element for the domain object. This value is not described in Change document. This value is not described for single property suggested by *multiple* attribute.

3.5.8 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 MUST 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 represents 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 represents 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 represents a title of the document.
- *count* attribute represents 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 represents 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.

3.5.9 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 MUST 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 represent a value of the property. These elements are defined in Section 2. 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 is regarded conjunctive.

- *type* attribute represents 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 represents 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 represents 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 represents 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 represents 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 represents that the property is expected to be calculated for the objects in the body of this document. For Get document, this attribute is used in *Selection* element. For Show document and Notify document, this attribute is described in *Header* element. This attribute does 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 represents 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.

4 Profile Specifications

4.1 Application profile Definitions

4.1.1 General

Application profile definition is a set of specifications for all application programs that may be involved in the communication exchanging PPS transaction messages. Each application program may send and receive messages that consist of domain documents, domain objects and domain properties. The application profile definition provides all available domain documents, domain objects and domain primitives.

Application programs can exchange their messages correctly when they understand the semantics of information in the message. In order to do this, application profile definition helps agreement of common usage and understanding of domain documents, domain objects and domain properties.

Several application profile definitions can exist independently for the same problem domain. Two application programs cannot communicate each other if they don't refer a common application profile. In order to avoid such a situation, this specification provides an extension mechanism in which a standard profile definition can be extended to an extended profile definition for particular group in local domain.

Figure 4.13 shows the structure of application profiles. Application profile is either a standard profile definition or an extended profile definition. Figure also shows that an implementation profile refers an application profile without regarding distinction of standard profile definition and extended profile definition.

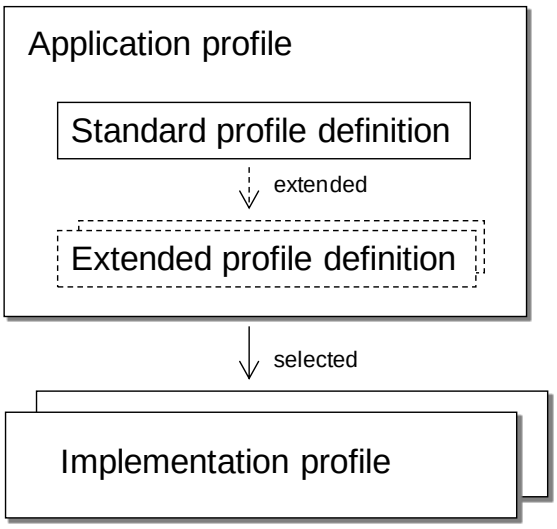


Figure 4.13 Structure of profile specifications

As an example of standard profile definition, PPS TC supports the PSLX profile [PROFILE] for this planning and scheduling domain. However, this specification only shows general rules and structures of a standard profile definition.

4.1.2 Structure of profile definitions

Application profile SHOULD have a list of domain documents and a list of domain objects. In addition, application profile MAY have a list of enumerations, which shows available value set of a domain property of a domain object.

Application profile definition SHOULD be described by *AppProfile* element defined in Section 4.3.1. This element SHOULD appear in the top level of the XML document.

All candidates of domain documents, which may be used by any application program who sends or receives a message in the target domain, SHOULD be specified using *AppDocument* element under the *AppProfile* element.

All domain objects, which are used in any domain document defined in *AppDocument* elements, SHOULD be specified in *AppObject* element under the *AppProfile* element. An *AppObject* has a list of properties that represent the characteristics of the object. Each property SHOULD be described in *AppProperty* under the *AppObject*.

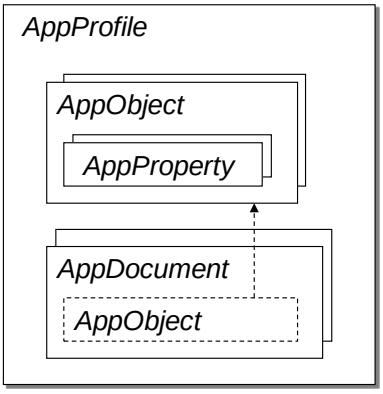


Figure 4.14 Application Profile

The structure of application profile is illustrated in Figure 4.14. Domain document represented by *AppDocument* has domain objects represented by *AppObject*. The domain objects that is listed in the same document SHOULD be the same class objects defined in one *AppObject* in the application profile. The application profile defines domain objects independent from domain documents, because the domain objects may be referred from several different kinds of domain documents.

Example: Application profile definition

```
<AppProfile name=”pps-profile” prefix="pps" namespace="http://www.oasis-
open.org/committees/pps/profile-1.0">
  <AppObject name="Product" primitive=”Item”>
    <AppProperty name="id" path="@id"/>
    <AppProperty name="name" path="@name"/>
    …
    <AppProperty name="Size" path="Spec[@type=”size”]/@value"/>
    <AppProperty name="Color" path="Spec[@type=”color”]/@value"/>
    …
  </AppObject>
  …
  <AppDocument name="ProductRecord" object=”Product”/>
  <AppDocument name="ProductInventory" object=”Product”/>
  <AppDocument name="BillOfMaterials" object=”Product”/>
  <AppDocument name="BillOfResources" object=”Product”/>
  …
</AppProfile>
```

4.1.3 Standard profile definitions

An application profile that does not have a base profile is a standard profile. Standard profile definition SHOULD be specified in consistent with the following rules:

- Standard profile definition SHOULD have a name to identify the definition among all application programs in world-wide. Unique identifier such as URI is required.
- The name of standard profile definition contains information of revision, and the revision of the definition SHOULD follow the rule defined in Section 4.1.5.
- Standard profile definition SHOULD NOT have a base definition as a reference of other standard profile definitions.
- Standard profile definition SHOULD be published among application programs and accessible by all the application programs in the problem domain via Internet by announcing the URL the application can download the document.
- Standard profile definition SHOULD have the domain object in Table 4.4 or sub-class of Table 4.1 domain objects. The domain objects SHOULD be represented by the primitive elements determined by the table.
- Every domain object in a standard profile definition SHOULD have a domain property that shows identifier of the object. The domain property SHOULD be represented by id attribute of the primitive XML element in Table 4.1.

Table 4.4 Domain objects required in standard profile definitions

Object Name	XML Element	Description
Party	*Party*	Party such as customers and suppliers
Plan	*Plan*	Plan of production, capacity, inventory, etc.
Order	*Order*	Request of products and services
Item	*Item*	Items to produce or consume
Resource	*Resource*	Production resource such as machine and personnel
Process	*Process*	Production process
Lot	*Lot*	Actual lots produced in the plant
Task	*Task*	Actual tasks on certain resources
Operation	*Operation*	Actual operations in the plant

4.1.4 Extended profile definitions

Standard profile definition MAY be extended by an extended profile definition. Extended profile definition MAY also be extended recursively. This is also represented by *AppProfile* element. Extended profile definitions SHOULD have a reference of a standard profile definition, which is the base of extension.

Extended profile definition MAY add domain documents, domain objects and domain properties which have not been defined in the standard profile definition. Additional information of domain documents, domain objects and domain properties SHOULD be defined in the same way as the definition in standard profile definitions.

Extended profile definitions MAY modify the domain documents, domain objects and domain properties addressed in the standard profile. In order to modify the definition, extended profile SHOULD describe new contents with the same identification name of the document, object or property.

Extended profile definitions SHOULD NOT remove the domain documents, domain objects and domain properties addressed in the standard profile.

Enumerations MAY be added or modified to the standard profile definition. When extended profile describes enumeration name which is in the standard profile, the candidates of the enumeration are

replaced to those in the standard. Extended profile definitions SHOULD NOT remove any enumeration in the application profile.

Example: Extended application profile

```
<AppProfile prefix="ex1" name="pps-profile-1.1" namespace="http://www.pslx.org/profile-1" base="pps-profile-1.0">
  <Enumeration name="groupType">
    <EnumElement name="high" description="description of a"/>
    <EnumElement name="low" description="description of b"/>
  </Enumeration>
  <AppObject name="Consumer">
    <AppProperty name="group" path="Spec[type='pslx:group']/@value" enumeration="groupType"/>
  </AppObject>
</AppProfile>
```

Example shows an application profile extended from the standard profile. The new profile has additional enumeration named "groupType", and then a new Consumer object is defined with a new property which has a name "group" and the additional enumeration type.

4.1.5 Revision rule

After an application profile definition has been created, many application programs are developed according to the profile definition. In accordance with the industrial experiences, the old definition may be required to modify for domain specific reasons in the application domain.

Any application profile SHOULD NOT be changed without keeping the following rules after when the profile definition has been published. Otherwise, the new profile SHOULD have a new name that doesn't have any relation with the previous one.

There are two revision levels. One is a revision that the system developers have to deal with the new specification and change if necessary. The other is editorial revision where the any program doesn't need to care in terms of interoperability. To inform the former cases, the name of profile SHOULD be changed by adding the revision numbers. For the latter cases, instead of changing the name of profile, the actual file name of the profile, specified at the *location* attribute in the *AppProfile* element SHOULD be changed.

In order to represent the revision status in the profile name, there are two portions of digits in the name of profile definitions: major revision and minor revision. They are following the original identification name or the profile separated by dash "-" mark. The two portion is separated by the dot "." character.

When the major version increases, it:

- SHOULD NOT change the name of the profile excepting the portion representing the revision status.
- SHOULD NOT change the prefix and namespece in the attribute of *AppProfile* element.
- SHOULD NOT change the domain object in *AppDocument* element.

When the minor version increases, it:

- SHOULD follow the rule of major version increasing,
- SHOULD NOT change the domain properties in the domain objects.
- SHOULD NOT change the enumeration definition in the *AppProfile* element.

4.2 Implementation profiles

4.2.1 General

Application program may not have all capability in dealing with the domain documents, domain objects and domain properties defined in the application profile definitions. Implementation profiles are the

selection of domain documents, domain objects and domain properties from application profile definitions by application programs depending on the capability of the program.

When an application program tries to send a message to another application program, system integrator may need to confirm whether or not the receiving application program has capability to response the message. Then an implementation profile of an application program shows such capability to send or receive information.

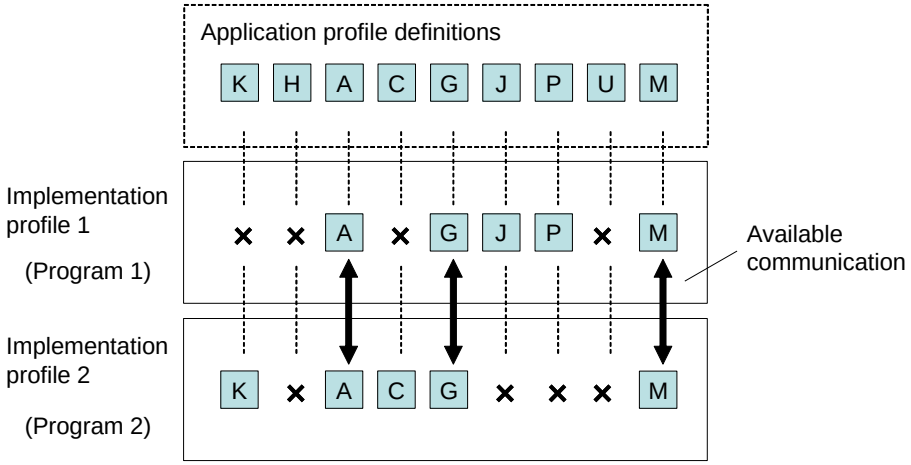


Figure 4.15 Concept of communication availability between implementations

Figure 4.15 explains a concept of communication availability between two application programs. Each application program that refers a same application profile has an implementation profile that has a list of items available to communicate, by selecting from the candidates defined in the application profile. Tow application programs can exchange a message properly if the both implementations have the corresponding capability.

An application program MAY have two or more than two implementation profiles each of which corresponding to different application profile definitions. An implementation profile SHOULD have a corresponding application profile definition.

To confirm the capability of any application program, section 4.2.4 provides the method of how to get the information by receiving an implementation profile from the program.

4.2.2 Structure of implementation profiles

Implementation profiles defined for application programs SHOULD be described by *ImplementProfile* element in XML format. The information includes domain documents, domain objects and domain properties available to process by the application program. For each domain document, implementation level, which shows the application program have all functions or not in terms of transactions defined in Section 3, can be defined.

Every implementation profile has a reference to a certain application profile. However, it doesn't show whether the application profile is a standard or extended. From the perspective of application programs, distinction between standard profile definition and extended profile definition has no sense.

ImplementProfile element MAY be described under *Transaction* element defined in Section 3. Therefore, this can be send or receive through a PPS transaction process. Using Get and Show transactions, two application programs can exchange the implementation profile.

An *ImplementProfile* element has *ImplementDocument* elements each of which represents availability for any domain document. An *ImplementDocument* element has *ImplementAction*, *ImplementProperty* and *ImplementEvent*.

ImplementAction element represents information of implemented type of transaction such as Get, Show, Add, and so forth. *ImplementProperty* element represents implemented properties of the domain object.

ImplementEvent represents any event definitions that the application program monitors properties and publish notifications of event defined on the property. Figure 4.16 shows the structure of *ImplementProfile*, *ImplementDocument*, *ImplementAction*, and *ImplementProperty* elements.

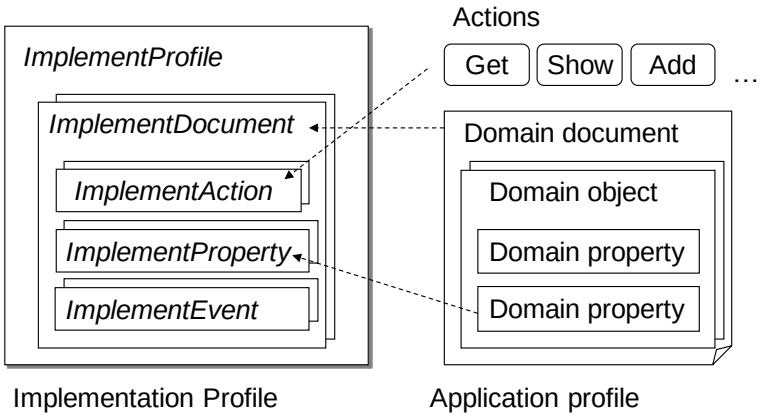


Figure 4.16 Structure of ImplementProfile

All domain documents represented by *ImmplementProfile* SHOULD be in the list of the corresponding application profile definition.

Domain documents in implementation profile SHOULD have a domain property if the property is defined in the application profile as a primary key of the object or as a property that is always required.

The following example shows an implementation profile of an application program that communicates with other program under an application profile. Then the implementation profile of the example is the selection of the application profile representing domain documents, transaction types and domain properties.

Example: Implementation profile of a program for an application profile

```
<ImplementProfile id=”AP001” action=”Notify”>
  <ImplementDocument name=”Product”>
    <ImplementAction action=”Get” level=”1”/>
    <ImplementAction action=”Show” level=”1”/>
    <ImplementAction action=”Add” level=”2”/>
    <ImplementProperty name=”id” title=”Company ID”/>
    <ImplementProperty name=”name” title=”Company name”/>
  </ImplementDocument>
  <ImplementDocument name=”ProductInventory”>
    …
  </ImplementDocument>
  ….
</ImplementProfile>
```

In accordance with the implementation profile, the application program sends or receives a message that SHOULD have a domain document listed in the implementation profile. The domain properties in the object SHOULD be one of the domain properties defined in the application profile.

Example: A message created on the implementation profile

```
<Document name=”Product” id=”001” action=”Get”
  namespace="http://www.oasis-open.org/committees/pps/profile-1.0">
  <Condition>
    <Property name=”pps:name” value=”MX-001”/>
    <Property name=”pps:color” value=”white”/>
  </Condition>
```

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

Above example shows a message of a Get document created by an application program. The properties referred to as “name” and “color” are specified in this message. The properties are defined in the implementation profile as well as the application profile. The prefix “pps” and colon mark are added at the front of the name to notify that the name is defined in the profile.

4.2.3 Level of implementation

Domain documents can be sent or received by application programs in any types of action including Add, Change, Remove, Get, Show, Notify and Sync. These actions are prescribed in Section 3. Level of implementation represents whether or not the functions prescribed in Section 3 are fully implemented or partially implemented

The certain level of Partial implementation is defined in Section 3 depending on the type of transaction. When the application program informs Partial implementation, it SHOULD have full capability of functions defined in the partial implementation in Section 3.

An application program MAY define a level of implementation for each pair of document and transaction type for each application profile definition.

4.2.4 Profile inquiry

All application programs SHOULD send implementation profile as a Show transaction message or Notify transaction message. Application programs SHOULD have capability to response implementation profile as Show message when it receives an *ImplementProfile* inquiry in a form of Get message.

When responding to the Get message of implementation profile in PULL model, the program SHOULD send corresponding Show message that is made of *ImplementProfile* element or *Error* element.

This capability of implement profile inquiry SHOULD NOT be in the available list of *ImplementProfile* by itself. Since any *Condition* and *Selection* element cannot be described in *ImplementProfile,* the inquiry of implementation profile can only request all the information of implement profiles.

Example: Inquiry of implementation profile for PPS standard profile definition

```
<Message id=”A01” sender=”A”>
  <ImplementProfile action=”Get” />
</Message>
```

Example: Answer of the inquiry in above example

```
<Message id=”B01” sender=”B”>
<ImplementProfile id=”B01” action=”Show” >
  <ImplementDocument name="Supplier">
    <ImplementAction action="Get" level="1"/>
    <ImplementAction action="Add"/>
    <ImplementProperty name="id" display="NO"/>
    <ImplementProperty name="name" display="NAME"/>
    …
  </ImplementDocument>

</ImplementProfile >
</Message>
```

Examples are the request of implementation profile and its response. By the message in the first example , the responder needs to answer its capability on the application profiles.

4.3 XML Elements

4.3.1 AppProfile Element

AppProfile element SHOULD represent an application profile. Standard application profile and extended application profile are both represented by this element. This is a top level element in an application profile, and has *Enumeration* element, *AppObject* element, and *AppDocument* 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="AppProfile">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Enumeration" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="AppObject" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="AppDocument" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="base" type="xsd:string"/>
    <xsd:attribute name="location" type="xsd:string"/>
    <xsd:attribute name="prefix" type="xsd:string"/>
    <xsd:attribute name="namespace" type="xsd:string"/>
    <xsd:attribute name="create" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *Enumeration* element SHOULD represent any enumeration type that is used as a special type of properties.
- *AppObject* element SHOULD represent any domain objects used in the domain documents defined in this profile.
- *AppDocument* element SHOULD represent any domain documents that the applications may send or receive on this profile.

- *name* attribute SHOULD represent the name of this application profile. The name SHOULD be unique in the namespace. This attribute is REQUIRED.
- *base* attribute SHOULD represent the base application profile when this profile is an extended application profile.
- *location* attribute SHOULD represent the location where the profile can be downloaded via Internet.
- *prefix* attribute SHOULD represent the prefix text that is added in the name of values that are qualified by this profile.
- *namespace* attribute SHOULD represent the namespace when this profile is used in a specific namespace.
- *create* attribute SHOULD represent the date of creation of the profile
- *description* attribute SHOULD represent any description related to this profile.

4.3.2 AppDocument Element

AppDocument element SHOULD represent a domain document that is contained in a message of any transactions. All domain documents that may appear in messages SHOULD be described in *AppApplication* element that corresponds to an application profile.

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="AppDocument">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="object" type="xsd:string"/>
    <xsd:attribute name="category" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *name* attribute SHOULD represent the name of the domain document. The name SHOULD be unique in the namespace to identify the type of the document. This attribute is REQUIRED.
- *object* attribute SHOULD represent the name of domain object that the document MAY have in the body as its content. One document SHOULD have one kind of domain object. All objects referred by this attribute SHOULD be defined in the same application profile or base application profile. This attribute is REQUIRED.
- *category* attribute SHOULD represent any category of the domain document. This information is used for making any group by categorizing various documents. Same group documents have same value for this attribute. This attribute is OPTIONAL.
- *description* attribute SHOULD represent any description of the domain document. Any comments and additional information of the document may be specified there. This attribute is OPTIONAL.

4.3.3 AppObject Element

AppObject element SHOULD represent a domain object corresponding to any actual object in the target problem domain. All domain objects that are referred to from domain documents in the application profile SHOULD be described in the *AppObject* 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="AppObject">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="AppProperty" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="primitive" type="xsd:string" use="required"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *AppProperty* element SHOULD represent a property that may be described in the domain objects of the application profile definition. All possible properties SHOULD be described in the domain object represented by *AppObject*.

- *name* attribute SHOULD represent the name of the object. The name SHOULD be unique under the application profile definition in the selected namespace. This attribute is REQUIRED.
- *primitive* attribute SHOULD represent a primitive element name selected from the primitive element list defined in Section 2. Since every domain object is a subclass of one in the primitive objects, all *AppObject* elements SHOULD have a primitive attribute. This attribute is REQUIRED.
- *description* attribute SHOULD represent any description of the domain object. This attribute is OPTIONAL.

4.3.4 AppProperty Element

AppProperty element SHOULD represent a domain property of a domain object. All properties that may be defined to represent the characteristics of the domain object SHOULD be described under the *AppObject* corresponding to the domain object.

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="AppProperty">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="path" type="xsd:string"/>
    <xsd:attribute name="multiple" type="xsd:string"/>
    <xsd:attribute name="key" type="xsd:string"/>
    <xsd:attribute name="enumeration" type="xsd:string"/>
    <xsd:attribute name="dataType" type="xsd:string"/>
    <xsd:attribute name="use" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *name* attribute SHOULD represent the name of the property. The name SHOULD be unique in the domain object defined by AppObject to identify the property. This attribute is REQUIRED.
- *path* attribute SHOULD represent the location of the attribute data in the primitive XML description defined in Section 2. The specification of the path SHOULD conform to [PATH]. If the profile is a standard application profile, this attribute is REQUIRED, and otherwise OPTIONAL.
- *multiple* attribute SHOULD represent whether the property can have multiple values or not. If the value of this attribute is positive integer or “Unbounded”, actual message described by Section 2 specification can have corresponding number of values for this property. This attribute is OPTIONAL.
- *key* attribute SHOULD represent whether or not this property is primary key of the domain object to identify the target object from the instances in the database. If the value is “True”, then this property is primary key. Primary key SHOULD NOT defined more than one in the same domain object.
- *enumeration* attribute SHOULD represent the name of enumeration type when the property has a value in the enumeration list. The name of enumeration type SHOULD be specified in *Enumeration* elements in the same application profile definition. This attribute is OPTIONAL.
- *dataType* attribute SHOULD represent the data type of the property. The value of this attribute SHOULD be “Qty”, “Char” or “Time”. The data type described in the attribute SHOULD be the same as the data type of attribute on the body elements identified by the path attribute.
- *use* attribute SHOULD represent that the property is mandatory for any implementation, if the value of this attribute is “Required”.
- *description* attribute SHOULD represent any description of the domain property. This attribute is OPTIONAL.

4.3.5 Enumeration Element

Enumeration element SHOULD represent an enumeration type that has several items in a list format. If a property of a domain object has the enumeration type, then the property SHOULD have one of any items in the enumeration list.

Enumeration type is independent from any domain object in the application profile definition. Therefore, several different domain objects MAY have different properties that has the same enumeration type.

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="Enumeration">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="EnumElement" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *EnumElement* element SHOULD represent an item of the list that the enumeration type has as candidates of property value.

- *name* attribute SHOULD represent a name of this enumeration type. The name SHOULD be unique in the application profile definition. This attribute is REQUIRED.
- *description* attribute SHOULD represent any description of the enumeration type. This attribute is OPTIONAL.

4.3.6 EnumElement Element

EnumElement element SHOULD represent an item of enumeration list. A property that is defined with the enumeration type SHOULD select one of the items from the enumeration list.

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="EnumElement">
  <xsd:complexType>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
    <xsd:attribute name="primary" type="xsd:boolean"/>
    <xsd:attribute name="alias" type="xsd:int"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *value* attribute SHOULD represent value texts that can be selected from the enumeration list. The value SHOULD be unique in the value list of the enumeration type. This attribute is REQUIRED.
- *primary* attribute SHOULD represent the primary item in the enumeration list. Only the primary attribute SHOULD have “True” value for this attribute. No more than one item in the item list SHOULD have “true” value. This attribute is OPTIONAL, and the default value is “False”.
- *alias* attribute SHOULD represent a numerical value instead of the text value specified in the *value* attribute. The value SHOULD be unique integer among the items in the enumeration type.
- *description* attribute SHOULD represent any description of the enumeration element. This attribute is OPTIONAL.

4.3.7 ImplementProfile Element

ImplementProfile element SHOULD represent an implementation profile for an application program. *ImplementProfile* SHOULD be defined for each application program what the application program supports. This information MAY be sent by the application program and received by the party who wants to know the capability of the application program. Therefore, in order to make transactions, some attributes and sub-elements are the same as the attributes of Document element defined in Section 3.

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="ImplementProfile">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Error" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="App" minOccurs="0"/>
      <xsd:element ref="ImplementDocument" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="action" type="xsd:string"/>
    <xsd:attribute name="profile" type="xsd:string"/>
    <xsd:attribute name="location" 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>
```

- *Error* element SHOULD represent error information, when any errors occur during the transaction of message exchange of this implementation profile. The specification of this element is defined in Section 3.
- *App* element SHOULD represent any information for the application program concerning the transaction of profile exchange. The use of this element SHOULD be consistent with all cases of transactions while the other messages are exchanged. The specification of this element is defined in Section 3.
- *ImplementDocument* element SHOULD represent a domain document that the application program may send or receive. All available documents in the application profile SHOULD be listed using this element.

- *id* attribute SHOULD represent identifier of the application program. The id SHOULD be unique in all application programs that can be accessed in the network. In order to guarantee the uniqueness, system integrator must assign the unique number and manages it in the network configuration. This id is the same as the sender name when the application will send a message. This attribute is REQUIRED.
- *name* attribute SHOULD represent a name that the application program shows its name for an explanation by natural texts. This attribute is OPTIONAL
- *action* attribute SHOULD represent a name of action during transaction models defined in Section 3. The value of this attribute SHOULD be “Notify”, “Get” or “Show”. When the element is created as a message for exchange, this attribute is REQUIRED. Otherwise, such as for a XML document file, this attribute is OPTIONAL.
- *profile* attribute SHOULD represent the name of application profile that this implementation profile is referring to select the available part in the definition. This attribute is OPTIONAL.
- *location* attribute SHOULD represent the location of the application profile to get the actual file by the party who want to know the content of the application profile. This attribute is OPTIONAL.
- *namespace* attribute SHOULD represent the namespace of the application profile. This attribute is necessary to identify the profile in world-wide basis. This attribute is OPTIONAL.
- *create* attribute SHOULD represent the date of creation of the implementation profile. This attribute is OPTIONAL.
- *description* attribute SHOULD represent any description of the implementation profile. This attribute is OPTIONAL.

4.3.8 ImplementDocument Element

ImplementDocument element SHOULD represent a domain document selected from the application profile. All available domain documents SHOULD be listed by this element. Available domain documents MAY be defined for each application profile that the program can support.

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="ImplementDocument">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ImplementAction" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="ImplementProperty" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="ImplementEvent" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="option" type="xsd:string"/>
    <xsd:attribute name="profile" type="xsd:string"/>
    <xsd:attribute name="location" type="xsd:string"/>
    <xsd:attribute name="namespace" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *ImplementAction* element SHOULD represent an action that the program can perform for the domain document. This element MAY represent a role of the program in the transaction.
- *ImplementProperty* element SHOULD represent a property that the program can deal with in the domain object. All properties defined in this element SHOULD be defined as a property of a domain object in the corresponding application profile.
- *ImplementEvent* element SHOULD represent an event that the program can monitor a property in order to notify the change of the data to subscribers. This information MAY be defined by each application programs.

- *name* attribute SHOULD represent the name of the domain document. The name SHOULD be defined in the list of domain document in the corresponding application profile. This attribute is REQUIRED.
- *title* attribute SHOULD represent the header title of the document. This value MAY be a short description to show the property relating to the actual world. This attribute is OPTIONAL.
- *option* attribute SHOULD represent optional process to deal with the domain document data. There can be several domain document of same document name if the document has different option value. According to the option process, the required implement properties may be different.
- *profile* attribute SHOULD represent the name of application profile that this *ImplementDocument* is referring to select the available part in the definition. This attribute is OPTIONAL.
- *location* attribute SHOULD represent the location of the application profile to get the actual file by the party who want to know the content of the application profile. This attribute is OPTIONAL.
- *namespace* attribute SHOULD represent the namespace of the *ImplementDocument*. This attribute is necessary to identify the document name in world-wide basis. This attribute is OPTIONAL.
- *description* attribute SHOULD represent any description of the implemented document. This attribute is OPTIONAL.

4.3.9 ImplementAction Element

ImplementAction element SHOULD represent an action that the program can perform for the domain document. The actions include the transaction model referred to as “Add”, “Change”, “Remove”, “Notify”, “Sync”, “Get” or “Show”. This element MAY represent a role of the program in the transaction such as sender or receiver.

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="ImplementAction">
  <xsd:complexType>
    <xsd:attribute name="action" type="xsd:string" use="required"/>
    <xsd:attribute name="level" type="xsd:int"/>
    <xsd:attribute name="role" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *action* attribute SHOULD represent the action performed by the application program. The value of this attribute SHOULD be one of “Add”, “Change”, “Remove”, “Notify”, “Sync”, “Get” and “Show”. This attribute is REQUIRED.
- *level* attribute SHOULD represent an implementation level defined in Section 3 for each document processed by the application program. Level 0 shows no implementation, while level 1 and 2 are partially and fully implemented, respectively. Default value is 1 that minimum implementation is supported. This attribute is OPTIONAL.
- *role* attribute SHOULD represent a role in the transaction. The value of this attribute is either “Server” or “Client”. Every transaction has its available roles that can be selected as a value of this attribute. Default value is “Server”. This attribute is OPTIONAL.
- *description* attribute SHOULD represent any description of the implement action. This attribute is OPTIONAL.

4.3.10 ImplementProperty Element

ImplementProperty element SHOULD represent a domain property that can be processed in the application program. Some properties SHOULD be defined in the corresponding domain object in the application profile definition. The properties that are not defined in the application profile SHOULD be specified in this element as a user extended property. Properties extended by application programs SHOULD have additional definitions similar to the definitions by *AppProperty* 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="ImplementProperty">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="extend" type="xsd:string"/>
    <xsd:attribute name="link" type="xsd:string"/>
    <xsd:attribute name="multiple" type="xsd:string"/>
    <xsd:attribute name="path" type="xsd:string"/>
    <xsd:attribute name="dataType" type="xsd:string"/>
    <xsd:attribute name="enumeration" type="xsd:string"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="use" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- *name* attribute SHOULD represent the name of the property. The name SHOULD be defined in the corresponding application profile. This attribute is REQUIRED.
- *title* attribute SHOULD represent the header title of the property. This value MAY be a short description to show the property relating to the actual world. This attribute is OPTIONAL.
- *extend* attribute SHOULD represent qualifier string that is specified as prefix of the property name, if this property is extended by the local program. For example, if the value is “user”, then the description of this property will have “user:” prefix in the actual messages. This attribute is OPTIONAL.
- *link* attribute SHOULD represent that this property is also defined in other domain document that can be linked to this document. The value of this attribute MAY has the name of domain document.
- *multiple* attribute SHOULD represent whether the property can have multiple values or not. If the value of this attribute is positive integer or “Unbounded”, actual message can have corresponding number of values for this property. The value number SHOULD be less or equal than the number defined in the application profile.
- *path* attribute SHOULD represent the location of the attribute data in the primitive XML description defined in Section 2. The specification of the path SHOULD conform to the syntax of [PATH]. If the attribute value of *extend* is defined and this attribute is not described, then the default path data SHOULD be “Spec[@type=’aaa:bbb’]/CCC/@value”, where aaa denotes the value of *extend* attribute and bbb denotes the value of *name* attribute, and CCC is the value of *dataType* attribute.
- *dataType* attribute SHOULD represent the data type of the property. The expecting value of this attribute is Qty, Char and Time. This attribute is REQUIRED if the value of *extend* has data. Otherwise it is OPTIONAL.
- *enumeration* attribute SHOULD represent the name of enumeration type when the property is extended by the local program, and has a value in the enumeration list. The name of enumeration type SHOULD be specified in *Enumeration* elements in the application profile definition. This attribute is OPTIONAL.
- *type* attribute SHOULD represent that the type of this property in terms of usage. When the value is “Typical”, then the usage of this property is typical.
- *use* attribute SHOULD whether the property is mandatory. When the value “Required” represents mandatory, while the value “Optional” represents optional. This value SHOULD be “Required” if the corresponding property in the application profile has “Required” value. Default value of this attribute is “Optional”.
- *description* attribute SHOULD represent any description of the property. This attribute is OPTIONAL.

4.3.11 ImplementEvent Element

ImplementEvent element SHOULD represent any event definitions that the application program monitors on a particular property and detects the event occurrence on it. When the event occurs, the application program SHOULD publish a notification of the event to all the parties who are on the list of subscription. This information is defined by each application program, then clients of the event notification service MAY request for the publication as a subscriber.

ImplementEvent element SHOULD allow an application program to define the unit size of data differences, maximum and minimum data value, duration of one monitoring cycle and expire date of notifications to determine the event occurrence.

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="ImplementEvent">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="App" minOccurs="0"/>
      <xsd:element ref="Condition" minOccurs="0" maxOccurs="unbounded"/>
```

```
        <xsd:element ref="Selection" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="type" type="xsd:string"/>
      <xsd:attribute name="cycle" type="xsd:duration"/>
      <xsd:attribute name="start" type="xsd:dateTime"/>
      <xsd:attribute name="expire" type="xsd:dateTime"/>
      <xsd:attribute name="description" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
```

- *App* element SHOULD represent the application specific information about event monitoring, event processing, transaction control and so forth. The specification of *App* element is defined in Section 2.
- *Condition* element SHOULD represent the condition to select the target domain objects the application is monitoring the event. The specification of this element is defined in Section 3.
- *Selection* element SHOULD represent the condition of selecting the target property in the domain object. The selected property values are reported to the subscribers when event occurs. When the target property is multiple, Condition element under this element can restrict the properties. The specification of this element is defined in Section 3.
- *Property* element SHOULD represent the target property and constraints to detect event on the property. The target property is monitored by the program. When there is more than one Property element under the *ImplementEvent*, it SHOULD represent that more than one conditions need to be checked to detect the event occurrence. Each Property element MAY have a different target property on the domain object to others. Conditions of these properties SHOULD be conjunctive. The specification of this element is defined in Section 3.

- *name* attribute SHOULD represent the name of the event. The name SHOULD be unique in the domain object defined in the application profile. This attribute is REQUIRED.
- *type* attribute SHOULD represent a method to detect this event. Value candidates of this attribute SHOULD include “True”, “False”, “Enter”, “Leave”, “Change”, “Add”, and “Remove”. If the value is “True”, then event occurs when all the conditions are true. If the value is “False”, then event occurs when at least one condition is false. If the value is “Enter”, then event occurs when the status changes from false to true, while “Leave” means that the status changes from true to false. If the value is “Change”, then event occurs when the value of the target property is change. “Add” represents that event occurs when a new domain object which satisfies the conditions is added, and “Remove” shows that event occurs when any objects which satisfies the conditions is removed. If the target property is multiple and *Selection* element is described, then “Add” and “Remove” mean that one of the multiple properties is added and removed, respectively. Default value is “Change”. This attribute is OPTIONAL.
- *cycle* attribute SHOULD represent the duration of monitoring of the property value to detect the event occurrence. The application program SHOULD monitor the value until the expiration date. This attribute is OPTIONAL.
- *start* attribute SHOULD represent starting time of the monitoring and notification service. After this date and time, application program start monitoring the properties. If this attribute is not described, then it represent the service has already started. The origin of cyclic procedure defined by cycle attribute SHOULD be this start time. This attribute is OPTIONAL.
- *expire* attribute SHOULD represent expire time and date of the event notification. After the time of expiration, the application will stop monitoring the event occurrence. If this attribute is not defined, it SHOULD represent that there is no expiration date. This attribute is OPTIONAL.
- *description* attribute SHOULD represent any description of the event. This attribute is OPTIONAL.

5 Conformance

A document or part of document conforms to OASIS PPS Core Elements if all elements in the artifact are consistent with the normative statements of Section 2 of this specification and the document can be processed properly with the XML schema that can be downloaded from the schema URI.

A document or message conforms to OASIS PPS Transaction Messages if all elements in the artifact are consistent with the normative statement of Section 3 of this specification and the document can be processed properly with the XML schema that can be downloaded from the schema URI.

A process or service conforms to OASIS PPS Transaction Messages if the process or service can deal with the message that conforms to OASIS PPS Transaction Messages and the process or service is consistent to the normative statement of Section 4 of this specification.

A document or profile conforms to OASIS PPS Profile Specifications if all elements in the artifact are consistent with the normative statements of this part of specifications and the document can be processed properly with the XML schema that can be downloaded from the schema URI.

The schema URI is given in the “Related work” section in the header page of this document.

A. Object Class diagram of Core Elements

Figure A.1 shows the structure of primitive objects in this specification with a UML class diagram. Each object corresponds to each XML element. In this figure, arrows represent relative information between the source and destination objects. When an arrow has role names, it corresponds to an independent XML element in the specification. This figure doesn't include all the information of XML schema but the partial information of the primitive elements.

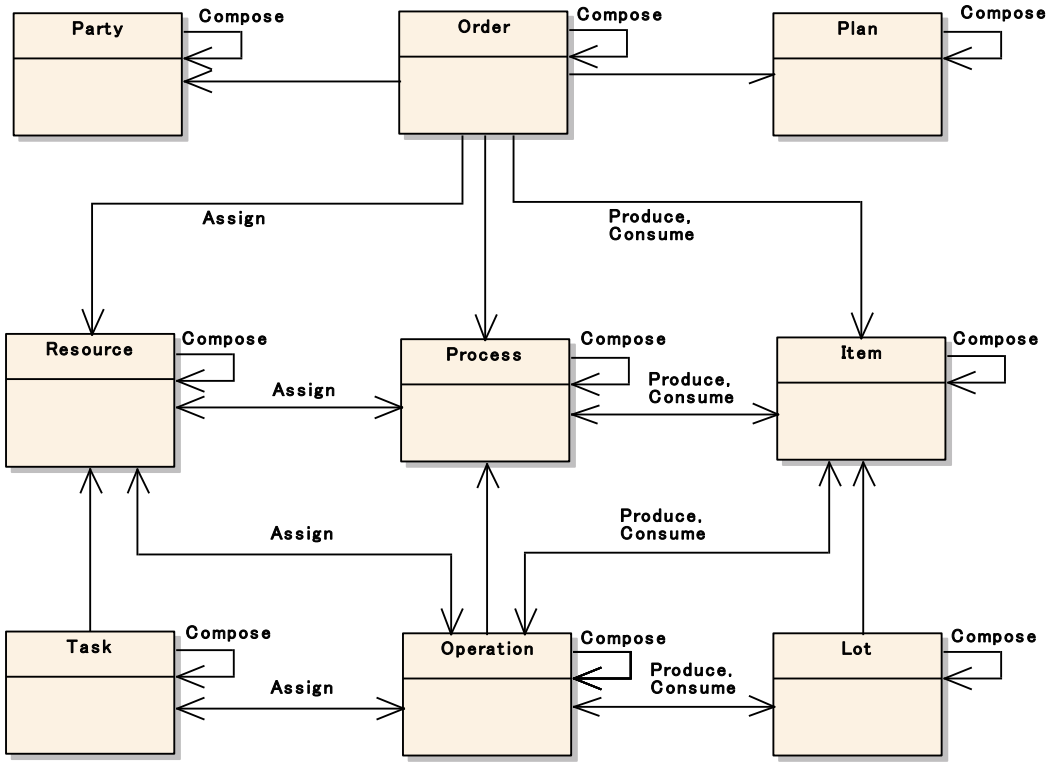


Figure A.1: Primitive objects for representing planning and scheduling problems

B. Cross reference of elements

Table B.1 shows the relations between elements. The row headers represent parent elements and the column headers represent child elements. Symbol * in the table means 0 or more than 0 element can be described.

Table B.1 Element and sub-element relations

	Compose	Produce	Consume	Assign	Relation	Location	Capacity	Progress	Spec	Start	End	Event	Price	Cost	Priority	Display	Description	Author	Date	Qty	Char	Time
Party	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Plan	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Order	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Item	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Resource	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Process	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Lot	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Task	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Operation	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Compose						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Produce						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Consume						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Assign						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Relation						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Location										*	*	*	*	*	*	*	*	*	*	*	*	*
Capacity										*	*	*	*	*	*	*	*	*	*	*	*	*
Progress										*	*	*	*	*	*	*	*	*	*	*	*	*
Spec										*	*	*	*	*	*	*	*	*	*	*	*	*
Start															*	*	*	*	*	*	*	*
End															*	*	*	*	*	*	*	*
Event															*	*	*	*	*	*	*	*
Price															*	*	*	*	*	*	*	*
Cost															*	*	*	*	*	*	*	*
Priority																				*	*	*
Display																				*	*	*
Description																				*	*	*
Author																				*	*	*
Date																				*	*	*
Qty																						
Char																						
Time																						

2822 The following table B.2 shows the correspondence between elements and attributes. The row headers
2823 show the element name, and the column headers show attribute the name. The characters in the table
2824 represent data types. The character in the table are used as follows: “U” denotes identification character
2825 of element, “P” denotes an identification character of referencing elements, “S” denotes the character
2826 string, “D” denotes a decimal number, “N” denotes an integer number and “T” for date time. Boldface
2827 means required information.

2828

2829 Table B.2 Element and attribute relations

	id	key	name	parent	type	status	apply	condition	value	count	unit	base	party	plan	order	item	resource	process	lot	task	operation
Party	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Plan	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Order	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Item	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Resource	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Process	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Lot	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Task	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Operation	**U**	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Compose	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Produce	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Consume	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Assign	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Relation	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Location	U	N	S		S	S	S														
Capacity	U	N	S		S	S	S														
Progress	U	N	S		S	S	S														
Spec	U	N	S		S	S	S														
Start	U	N	S		S	S	S	S	S												
End	U	N	S		S	S	S	S	S												
Event	U	N	S		S	S	S	S	S												
Price	U	N	S		S	S	S	S	S												
Cost	U	N	S		S	S	S	S	S												
Priority			S		S	S	S	S	S												
Display			S		S	S	S	S	S												
Description			S		S	S	S	S	S												
Author			S		S	S	S	S	S												
Date			S		S	S	S	S	S												
Qty			S		S	S	S	S	D	N	S	D									
Char			S		S	S	S	S	S	N	S	S									
Time			S		S	S	S	S	T	N	S	T									

2830

C. Implementation level

Since this specification provides the highest level functionality of application programs of information exchange on planning and scheduling problems, it might be difficult 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 application program.

The implementation level is specified in implementation profiles defined in Section 4. 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 C.1 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" denotes that the section or subsection is not necessary for the application program that declares level 1 for the messaging model.

D. Revision History

Revision	Date	Editor	Changes Made
01	23 Feb 2011	Y.Nishioka	Marge three parts of CS01
02	24 May 2011	Y.Nishioka	Name space URI and Cover page URI are revised

E. Acknowledgements

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

Participants:

Yasuyuki Nishioka, PSLX Forum/Hosei University
Koichi Wada, PSLX Forum
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
Tomoichi Sato, Individual
Junzo Kato, PSLX Forum
Hiroaki Machida, PSLX Forum
Shoei Komatsu, PSLX Forum