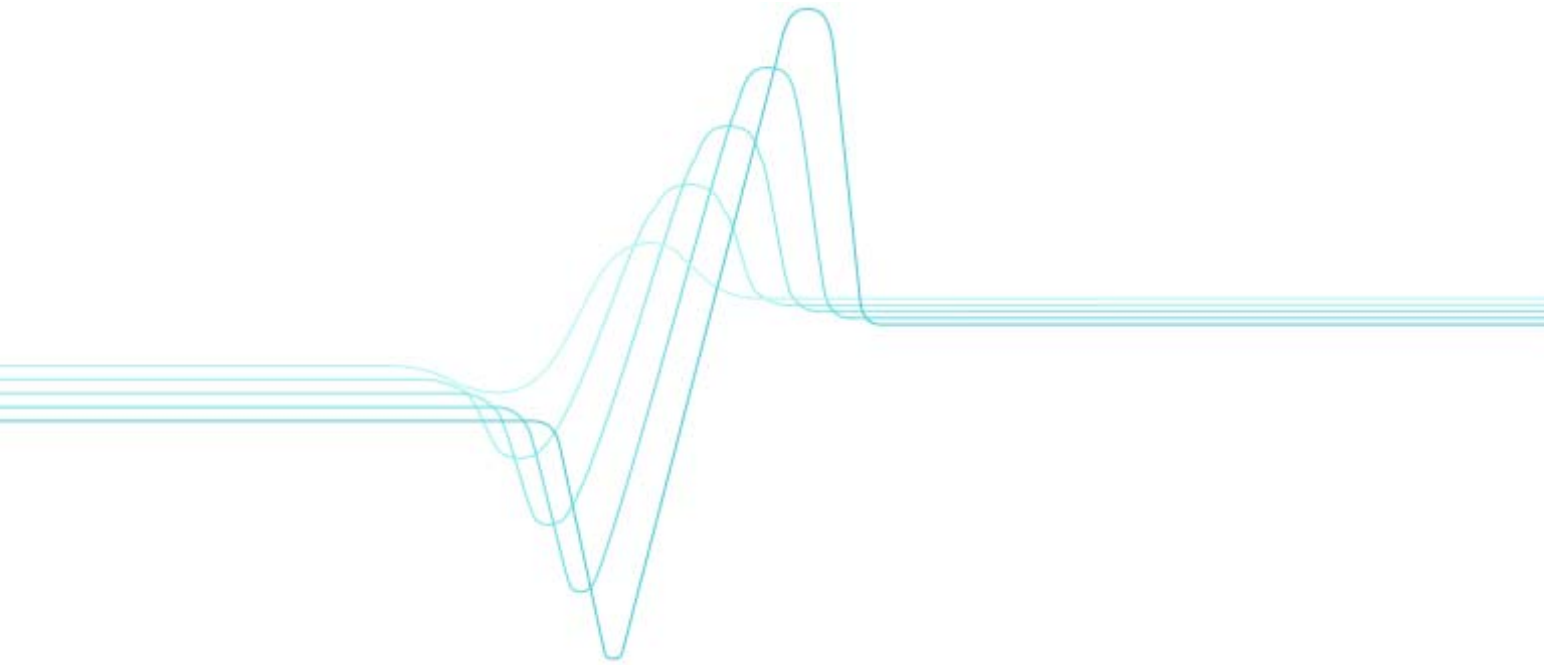
ESPOO 2004 VTT RESEARCH NOTES 2277

Kalle Kondelin, Tommi Karhela & Pasi Laakso

Service Framework Specification for Process Plant Lifecycle



VTT TIEDOTTEITA – RESEARCH NOTES 2277

Service Framework Specification for Process Plant Lifecycle

Kalle Kondelin, Tommi Karhela & Pasi Laakso

VTT Industrial Systems

ISBN 951–38–6521–5 (soft back ed.)
ISSN 1235–0605 (soft back ed.)

ISBN 951–38–6522–3 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0865 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 4374

VTT Tuotteet ja tuotanto, Tekniikantie 12, PL 1301, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 6752

VTT Industriella System, Teknikvägen 12, PB 1301, 02044 VTT
tel. växel 020 722 111, fax 020 722 6752

VTT Industrial Systems, Tekniikantie 12, P.O.Box 1301, FIN–02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 6752

Otamedia Oy, Espoo 2004

Kondelin, Kalle, Karhela, Tommi & Laakso, Pasi. Service Framework Specification for Process Plant Lifecycle. Espoo 2004. VTT Tiedotteita – Research Notes 2277. 123 p.

Keywords industrial plants, industrial processes, plant models, value-added services, web services, service networks, life cycle evaluation, maintenance, simulation, information management

Abstract

This specification descibes a software infrastructure for storing, deploying and using process plant information through the lifecycle of the process plant from design to construction and usage. Document describes the architecture and interfaces of the infrastucture. The specification was written in a research project (2003–2004) funded by National Technology Agency (Tekes), Finnish industry and VTT. This document is one of the end products of the ProServ project.

Contents

1. Introduction

1.1 Background and Motivation

Around the world the networking trend is increasing. Business partners come quickly and easily together to benefit from a business opportunity, fulfil the business need and collaborate. Life cycle considerations have enlarged this approach from production and shop-floor activities to innovative early stages of product lives like engineering and design, and to later stages in customer collaboration, like service and maintenance. In addition to industry, similar trends can also be seen in public and non-profit organizations.

Open extensibility for value-added services are becoming key issues in information management related to process plants, their equipment and automation. The progress in software technologies offers possibilities for new kind of integration of process design and simulation tools as well as value-added services for process operation and maintenance. In order to achieve this kind of integration, a domain specific specification of a service framework is needed. The specification should take into account the software architectural aspects and the fact that the process designers, operators and other end users are working in distributed enterprises using several existing production related information systems.

As compared to the use of separate systems per se, the proposed service framework provides a common user interface and knowledge presentation, and a common way to extend the existing systems with new (best of the breed) value-added services. The new value-added services will be implemented as web services, published by service providers in a common service interface repository. The framework also enables discovering and binding of the services by service requestors. It thus makes it viable to construct, compose and consume software component based web services, which will add domain specific value.

Life cycle information management is a concern in many research and development activities today. Life cycle information management has been analyzed in a three years research project, NIITM at VTT Industrial Systems (VTT) and at Tampere University of Technology (TUT). Process modeling and simulation in distributed company networks has been analyzed in a three years research project Gallery at VTT, at Helsinki University of Technology (HUT) and at Lappeenranta University of Technology (LUT). All these activities support the need of a generic framework for information management.

1.1.1 Need for Common Data Model

Efficient networking emphasizes the need to have a common data model. In different phases of the life cycle several interest groups interact and need to exchange information between each other. Also the increasing demand on the speed have enlarged the need of parallel activities in all phases and, consequently, also the need of distribution of information. However, especially in the early stages of the life cycle a lot of information is sensitive. It may deal with innovative, emerging knowledge. It is usually neither complete nor is the documentation available. Knowledge and information management becomes an important issue, which has to rely on data models.

During the realization phase, the plant gets its final form, when a lot of technical information may also be specified and updated and changes take place. Project management itself is supported by several tools that schedule projects, map their risks, keep project organization charts etc. The need for shorter delivery times and more successful projects challenge the interest groups that take part to big one-of-a-kind projects, like process plant delivery projects, to integrate the project management issues more tightly with the technical plant information. Further, the hand-over of the plant from the project organization to the organization that operates it, requires binding together the process design, automation design and project management information of the entire delivery project. This ensures that when the plant hand-over takes place, all technical information and manuals can simultaneously be provided to the operator (available in e.g in electronical form), since this information is taken care of during the project in proper manner. Often, however, immediately after the project, the information exchange between the deliverers and operator of the plant is inadequate, which hampers starting-up the production in the plant; all necessary information for (effective) use of the plant is not available.

To reach the objectives mentioned above, a proactive approach is required: knowledge and information from previous projects must be utilized systematically. The idea is depicted in *Figure 1-1*.

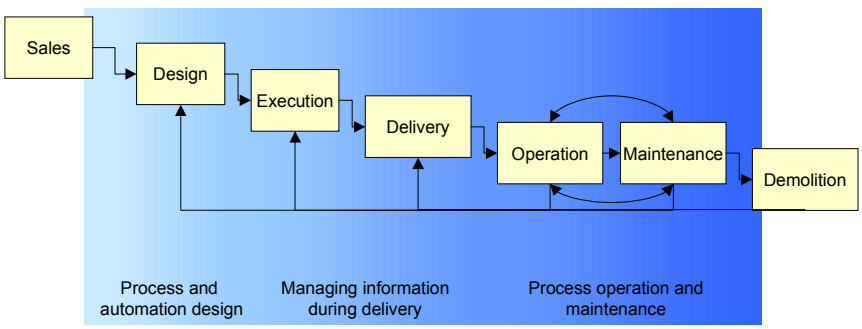


Figure 1-1. Information utilization in process plant lifecycle.

Proactivity requires managing information down- and upstream – use and reuse of information is essential. In many reported cases the main emphasis has so far been on collection and transmission of operational data. A huge task is still the analysis and resorting of the collected data in order to make it usable for diagnostics, design and engineering. The intended use of information will have a significant impact on the methods needed for this prescreening.

A common data model offers possibility for transformations between different application software attached to the framework. For example different management, design and simulation tools are used at different stages of the process life cycle and many of them are incompatible. So far the transition from one stage to another requires re-configuration for the specific tools. By conforming to a common data model and using transformations through this common data model at least some of the re-configuration can be avoided.

1.1.2 Model based process and automation design

While software technologies are developing further, software architectures are becoming more open and more extensible. This is also a trend in simulation. CAPE-Open and HLA (High Level Architecture) specifications are examples of such a development. These specifications are, however, very model-centric. They focus on independent development of model blocks that can be attached to the execution of the simulation engine or run-time infrastructure.

On the other hand, there are also more data-centric standardization efforts e.g. STEP (ISO 10303 parts 221, 227, 231) and PDML (Product Data Markup Language). The data-centric specifications are often large and concentrate mainly on conceptual design and product data management rather than on architectural issues.

Third group of requirements is for model usage. Simulation architectures provide open interfaces e.g. for data access, historical data access, simulation and training control. An example of popularly used standards is OPC (OLE for Process Control).

For process simulation, integration means reuse of model configurations and parameter values, co-use of different simulation and other process design tools, easy extensibility of simulator functionality and closer interoperability between distributed control systems and dynamic process simulators. The framework specification should take into account and combine the requirements for model development, model configuration and model usage. Easy extensibility for the framework means that the value-added services such as dimensioning tools, transformers between different simulators and external

simulation models can be developed to the framework in a flexible way. The same applies to non-simulation based design tools. The extensibility of the framework enables adding transformers between design tools and information transformations between technical design information and project management information.

It would be useful, e.g., for equipment manufacturers to publish information on their equipment in the form of models. The model software components could be transmitted in the framework to the designers and end users and the models could be used as part of different process simulation software. This is possible when using standardised interfaces between external models and simulation engines (e.g. CAPE-Open).

1.1.3 Managing information during delivery

Efficient information management during plant delivery is fundamental for the realization of the plant. The project-organization (delivering the plant) consists of many different partners with very different information needs and different ability e.g to utilize information systems. However, they should all have access to information, which they need to perform their tasks efficiently. During this phase the plans from the design and engineering phase are realized and e.g. final specifications and component decisions made.

At the moment when the plant is taken over by the operator and set to the productive use, the users must have access to all relevant information in a form adapted to their needs. Process information, information on the structures that conducts the desired processes and information about the use of the plant are typically information that is needed in the start-up phase and onwards.

Practically, this information needs to be gained from different interest groups of the plant project and communicated from the project organization to the operator. Several interest groups may also use different tools in their jobs and this further hampers the information management.

The framework supports managing information between planning/design phase and operation by:

- Providing interfaces for design tools in planning and design so that adequate information can be gathered and in fact is gathered if the design tools utilize the interfaces that the framework provides. This information is further utilized later during operation and maintenance. *Figure 1-2* depicts the process.

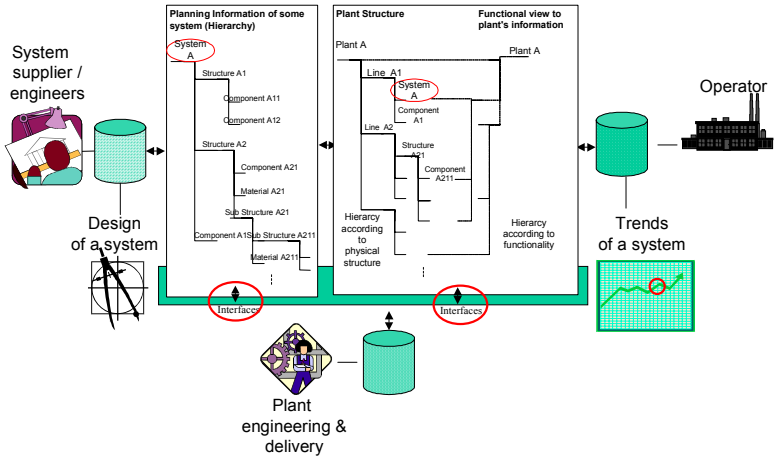


Figure 1-2. Gathering the information during design-phase and utilizing it during operation & maintenance. During delivery, the information is refined and communicated.

- Providing interfaces, which allows partners in the delivery project to efficiently exchange information about the plant and to submit new information to the plant database, when this is needed.
- Providing interfaces for tools in operation and maintenance so that adequate information can be gathered. Communicating this information upstream enables proactivity between subsequent projects – and also plants.
- Open extensibility that enables using transformers etc. that communicate information between the (tools of) design and operation phases – in other words also over the lifecycle in addition to communication between the interest groups during design and during operation.
- Supporting linking the information management to project management process: information is really produced on time etc.

In defining and implementing the framework in all lifecycle phases, the need to communication the information also between the lifecycle phases must be kept in mind, so that the framework supports that too. This may have effects on the definition of the framework also within a lifecycle phase (like design, operation & maintenance), since already here it must be thought of, what information is needed in the other phases of the lifecycle.

1.1.4 Process operation and maintenance

The operational phase of a plant is the longest part of its lifetime. During this period the plant is performing its planned tasks, when also all costs during its lifetime must be paid back. Naturally, efficient and smooth operation with little disturbances is a main target

for its owner. The idea of value-added services during process operation and maintenance is to ease various tasks in the lifecycle of a plant or transitions from one lifecycle phase to another. The value-added services accomplish this by enabling the use of previously produced information and experiences:

- From the same project or plant
- From other related projects and plants.

Information from existing systems, including automation, maintenance, production control, and condition monitoring systems, is analyzed in order to develop new, value-added services. These services are then embedded in the service framework.

The value-added services enable existing information systems to collaborate so that information flows related to maintenance are improved. Services may both read and produce information from and to existing systems so that e.g. state of a process can be read from a condition monitoring system and new work orders can be produced to a maintenance system by a service. In this way, the framework serves various co-operating interest groups.

For example following value-added services are needed:

1.1.4.1 Analysis of equipment use for maintenance

More precise analysis of use and data produced by automation systems to plan maintenance so that during maintenance break also those systems and machines that send weak signals of reduced capacity can be maintained. If not maintained during maintenance break, they may later require an additional break for maintenance. Realizing this value-added service requires transferring know-how from suppliers to the system, so that the system can recognize trends that point to a hidden malfunction that can later cause another production break. The service is depicted in *Figure 1-3*.

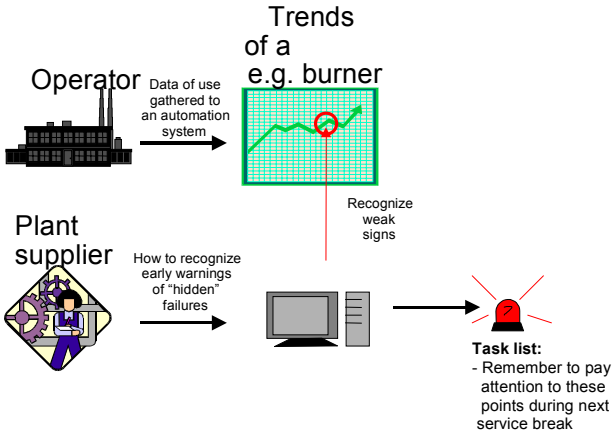


Figure 1-3. Analysis of data from use to detect the early warning signs.

1.1.4.2 Direct the costs of a system failure to a structure that caused the failure

During NIITM project's case studies it was expressed that the financial consequences of malfunctions and additional production stops caused by machine or system failures are not linked to the structure that caused the losses. Directing the costs to the troublesome structure would provide a possibility for later analysis of structures and their degree of quality. The idea is clarified in *Figure 1-4*.

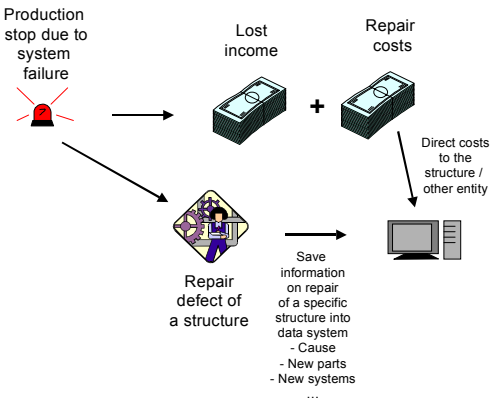


Figure 1-4. Costs caused by a production stop due to a malfunction of a device.

In practice the willingness to support value-added services means that when defining the framework, the interfaces that enable adding the services later are included in the definition. For process operation and maintenance services, integration means mechanisms that make the information systems for process management, process control, condition monitoring and maintenance interoperable with each other. This interoperability serves process operators, maintenance personnel (whether in-house or outsourced) and production supervisors as well as system suppliers and service providers. Easy extensibility for the framework means that the value-added services exemplified above can be embedded into the framework in a flexible way.

1.2 The Objectives

The objectives of this document is to define a common framework specification for process plant information management in order to support different phases of plant life cycle and open extensibility for value-added services in a distributed working environment consisting of different companies. The framework specification will focus both on vocabulary (data model) and architectural issues. The framework has three main design premises: model based process and automation design, process information management during delivery, operation and maintenance.

The goal is to improve information management in the process plant life cycle by specifying a common framework for different legacy tools used in different phases of plant delivery in various companies. In the framework definition the emphasis is on simulation assisted process design, process information management during the delivery and process operation related maintenance. The integration of information flows and sharing of information between the mentioned life-cycle phases are the key issues.

1.3 Structure of the Document

ProServ framework is a software infrastructure for storing, deploying and using process plant information through the process lifecycle. This document describes the software architechture following loosly the spirit and practice recommended by IEEE 1471 (IEEE 2000).

The minimun set of stakeholders according to the standard are users, acquirers, developers and maintainers. The users are further divided into software model designers, model type designers, model configurators, and model users.

- Model type designer designes a new (plant object) type hierarchy and rule set for it.
- Paradigm designer designes a new paradigm, an old paradigm to a new type hierarchy, or a new calculation component to an existing paradigm for example using a different algorithm.
- Provider (e.g. equipment manufacturer) provides model templates and process component data for the model configurators.
- Model configurator (e.g process and/or automation designer) designs a new plant model using existing types and templates.
- Model user (e.g. operator) uses a model made by model configurators.

- Acquirer (e.g ICT manager) is involed in taking the framework in use for a company taking part to the life-cycle of the process plant.
- Developer is involed in developing the framework.
- Maintainer is responsible for maintaining the framework operational and up to date.

The selected viewpoints are logical, security, component, deployment, and interface. Each selected viewpoint is represented by a chapter in this document.

The logical viewpoint describes the key concepts of the framework. Services of the framework are described using these concepts and their relations. Together with the use case analysis this view describes the functionality of the framework. The use case analysis is described in the use case analysis document (Salkari et al. 2004). This viewpoint is important for all stakeholders.

The security viewpoint describes the security model of the framework. In a distributed, multi-user, multi-company environment the security concerns are of the utmost importance during the process plant life cycle. This viewpoint is important for all stakeholders.

The component viewpoint describes the framework software components and their relationships. The framework is broken into sub-systems, packages and libraries in the way they are used in the sowftware development environment. This viewpoint serves the needs of the developers and maintainers.

The deployment viewpoint describes the physical architechture of the framework. The framework can have different deployments in different cases, each deployment using certain features of the framework. By going through few sample examples the idea behind the framework is shown. This viewpoint serves the needs of the acquirers, developers and maintainers.

The interface viewpoint descirbes the interfaces of the architecture and the data model behind the interfaces. This viewpoint serves the needs of the developers and maintainers.

2. Logical View

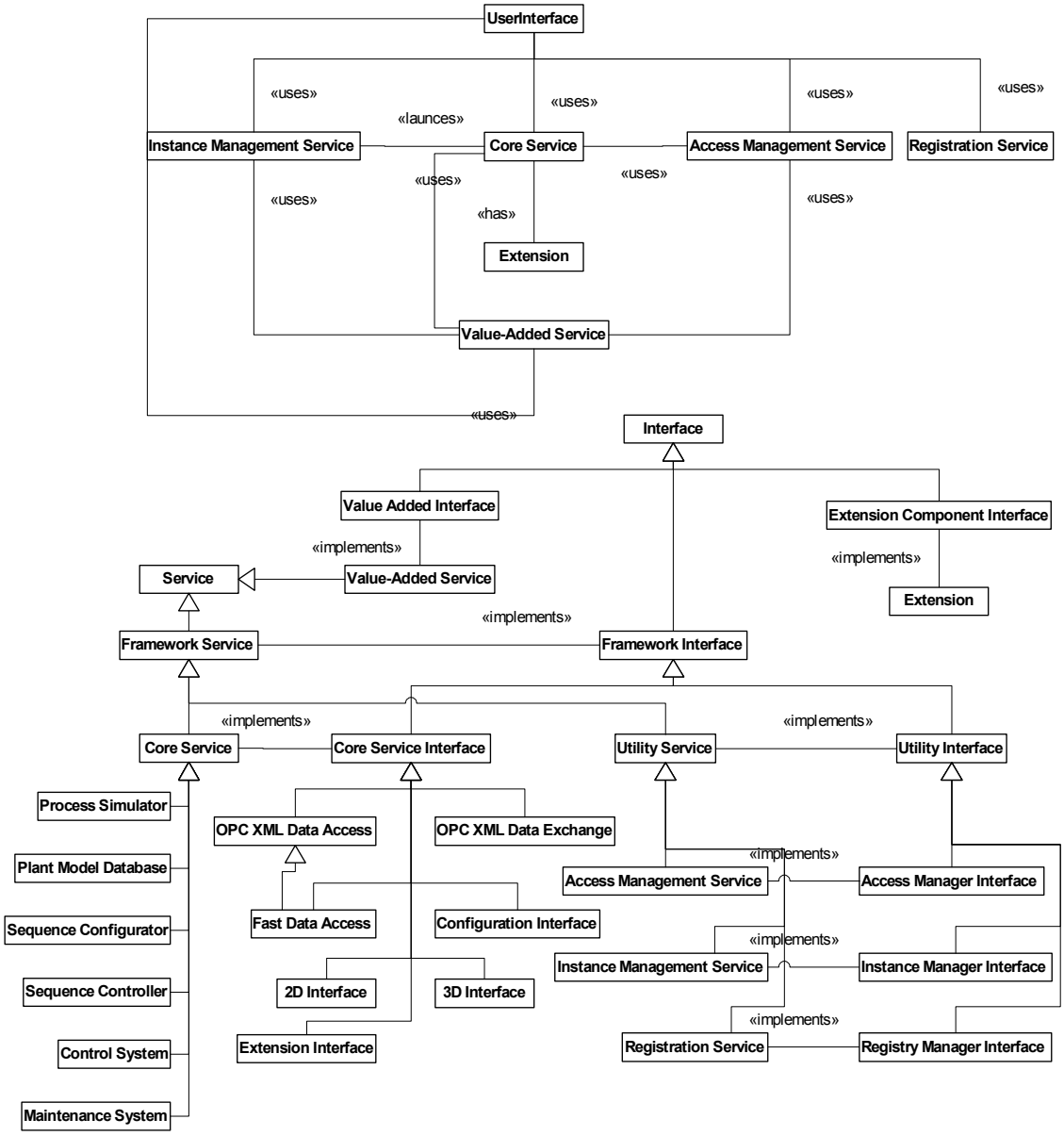


Figure 2-1. Logical view.

The logical view to the framework is described in *Figure 2-1* as UML calss diagram. The architecture consists of services. The services can be classified as framework services and value-added services. Framework services can further be classified as utility services and core services. Access management, instance management, and registration services are utility services. Core services are the services containing the base functionality of the life-cyle information management of a process plant, i.e. the

main focus of the framework. For example control systems, maintenance systems, process simulators and design databases can be considered as core services in the framework. The value-added services are the services built on top of the framework, providing added value to clients.

Registration service provides information about other services. It is a yellow pages type of service, used for querying what services there are, and what their properties are. Instance management service provides mechanisms to start services and information about the URIs of the started services. This is for those services that are not always running, and that do not have fixed URIs. Access management service provides user management and authentication services. Core services provide process plant model information in various forms, and value-added services provide case specific web services added on top of the framework.

Each service in the framework has an interface. Interface is a description of the way the service can be accessed. Except for the extension component interface, all interfaces are web service interfaces, and their normative descritption is written in WSDL (W3C 2001). Extension component interface is the interface that an extension component must implement. An extension component is a software component inserted into the plant model of a core service by some client, and it extends the functionality of the model, e.g. a calculation model of a plant object can be inserted as an extension component to the core service.

Configuration, extension, 2d, 3d, OPC XML data access, fast data access, and OPC XML data exchange are the core service interfaces. Each core service implements one or more of these interfaces. Configuration interface is for accessing the toplogial configuration of the plant. Extension interface is for calculating different details in the plant model, e.g. simulating the dynamic behaviour of the plant. 2d interface can be used for configurating and accessing two dimensional views to the plant model. 3d interface can be used for configurating and accessing three dimensional views to the plant model. OPC XML data access, fast data access and OPC XML data exchange interfaces are for accessing the time dependent process data, either from the real plant or from virtual plant model.

3. Security View

The security model is based on Web Services Security Model (Oasis 2002). The security model of the framework can be divided into three parts: authentication, encryption, and access protection.

The clients and servers can be authenticated using signed security tokens passed in the SOAP (W3C 2003) headers. The signing is based on XML Signature specification (W3C 2002b) and the authentication is based on password or public key.

The SOAP messages passed between clients and servers can contain sensitive information, and to hide them from potential eavesdroppers, the SOAP messages or parts of them can be encrypted using methods described in XML Encryption (W3C 2002a) specification.

The third part of the security is the access model built on the data model. Each element derived from ElementA has access rights, and can be accessed only by users having proper access rights. The users and groups (which are collections of users) are managed by the access management service, a web service whose interface is described in interface view. The access to an element and its data is based solely on the access rights of the given user (each request is done with the credentials of some user). The place of the element in the hierarchy of elements does not contribute anything to the access rights, unlike in the typical directory/file system security models.

4. Component View

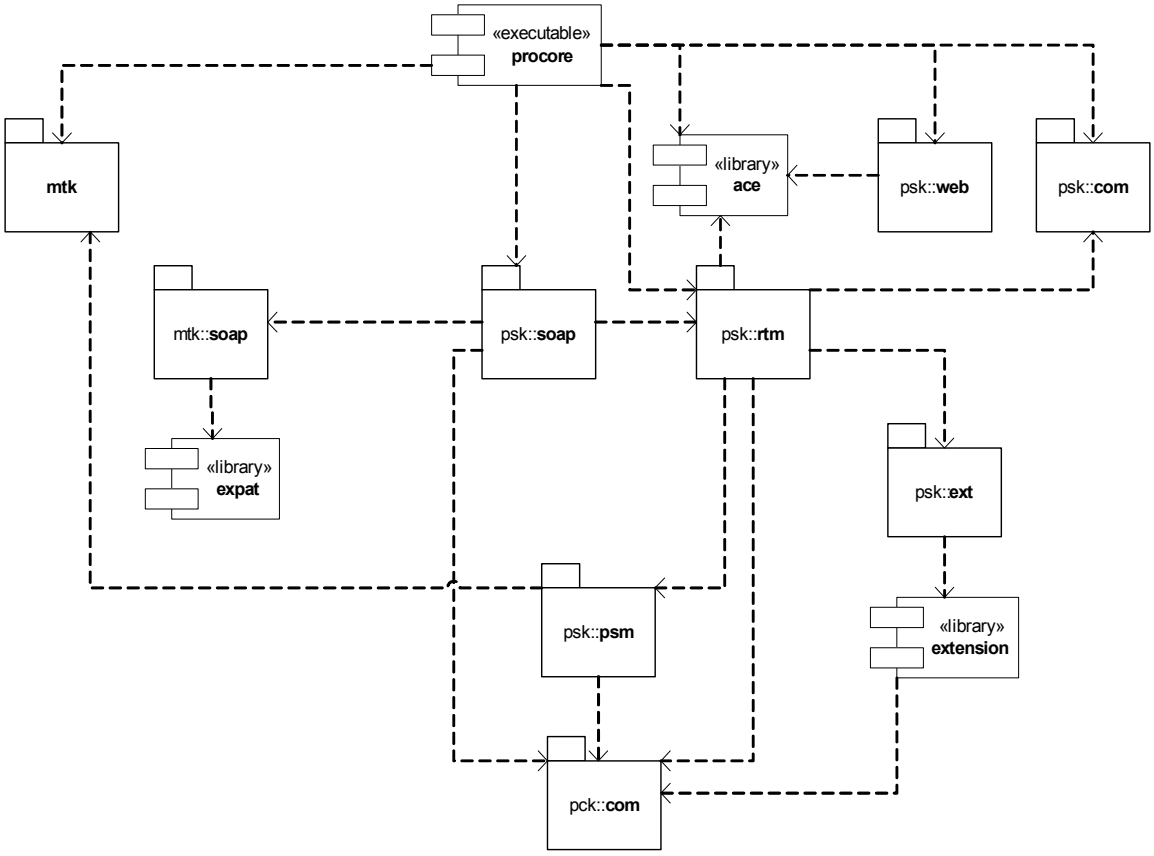


Figure 4-1. Components of procore.

The prototype implementation of the framework is made of software components, proconf and procore. Proconf is a user interface for framework services, and procore is a framework service implementing core and utility service interfaces. The component view of the procore as UML diagram is shown in *Figure 4-1*. The procore executable is made of shared libraries of Expat (Expat 1999), ACE (ACE 2004) and extension components. Expat is a sax XML (W3C 2000) parser, ACE is an Adaptive Communication Environment, and extension components are libraries containing calculation models related to plant objects.

Procore is made of packages, which are utility toolkit (mtk), SOAP toolkit (mtk.soap, psk.soap), common definition package (pck.com, psk.com), extension management (psk.ext), persistency management (psk.psm), runtime management (psk.rtm), and web service management (psk.web).

Utility toolkit implements few utility classes, most importantly allocation and release of persistent memory pages. SOAP toolkit implements classes for demarshalling the SOAP

requests and marshalling the SOAP responses. Common definition package implements common datatypes and classes for other packages. Extension management wraps the function interface of extension components to a class interface. Persistency management implements the persistency of the plant model data. Runtime management implements the runtime functionality of calculation models and data exchange. Web service management implements the marshalling and demarshalling of HTTP (W3C 1997) protocol requests and responses.

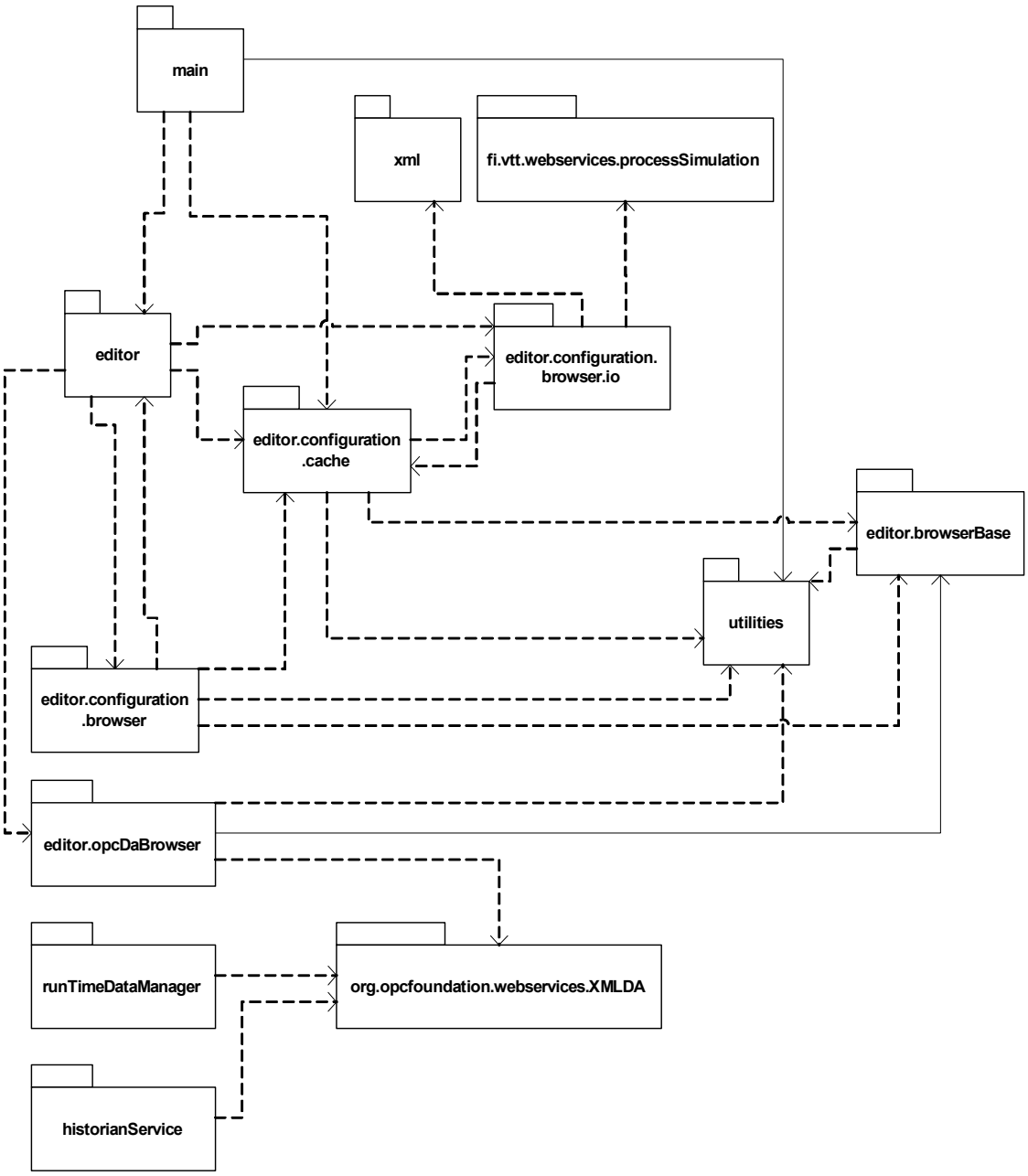


Figure 4-2. Packages of proconf.

The packages of the proconf as UML diagram are shown in *Figure 4-2*. Main package implements the main procedure of the program and handling of the main window frame. Editor package implements the sub frames of the main frame. These include e.g. the browse view and message view frames. Utilities package implements some utility and global information classes used in many parts of the proconf implementation. The handling of xml is separated in two packages called xml and fi.vtt.webservices.processSimulation. The xml package handles the locally stored xml files. The transformation from xml to java classes is based on tool called Castor (Castor 2004). fi.vtt.webservices.processSimulation handles the web service communication with the procore. The messages are interpreted using the Axis package (Apache 2004). editor.configuration.browser.io package takes care of the communication between the proconf and xml packages.

editor.configuration.browser (called Browser package later) contains all the browser and browser node classes. editor.configuration.cache (called Cache package later) contains the cached information received from the xml packages. The cache objects provide subcribe interface, which the browser package uses to subscribe to get information about the changes in the cache. editor.browserBase contains classes, which are used as base class for classes presented in the Browser and Cache packages. Furthemore it contains some utility classes typically needed for implementing treeviews.

editor.opcDaBrowser contains the classes needed to show the contents of the OPC DA server and communicate with the OPC server using the browse interface of the OPC DA server.

RuntimeDataManager includes the classes and interface, which wrap the functionlity provided by the OPC XML DA server. org.opcfoundation.webservices.XMLDA contains classes generated using Axis tool, which interpret the web service messages provided by the OPC XML DA servers.

HistorianService contains interface and classes used to read and write time series data.

5. Deployment View

5.1 Simulation assisted control system testing

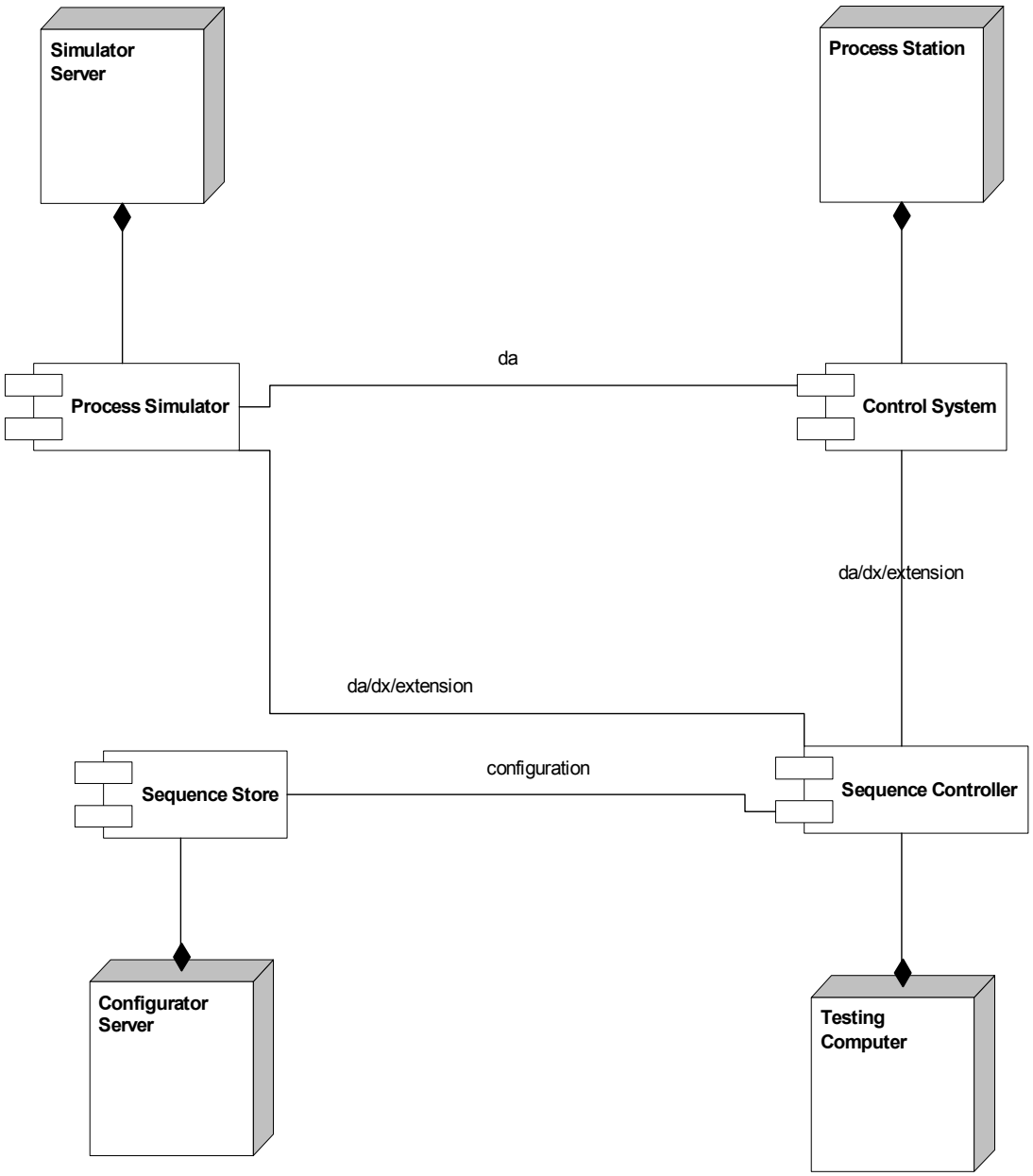


Figure 5-1. Deployment in the automation desing test case.

In this case the objective of the deployment is to enable automatic testing of control solution against a simulator. The physical deployment of the case is shown in *Figure 5-1*. In the case there are three core services, process simulator, control system, and sequence store. In this deployment all core services are in their own nodes. In addition

to the core services there is sequence controller, a value-added service using the three core services to implement the automatic testing.

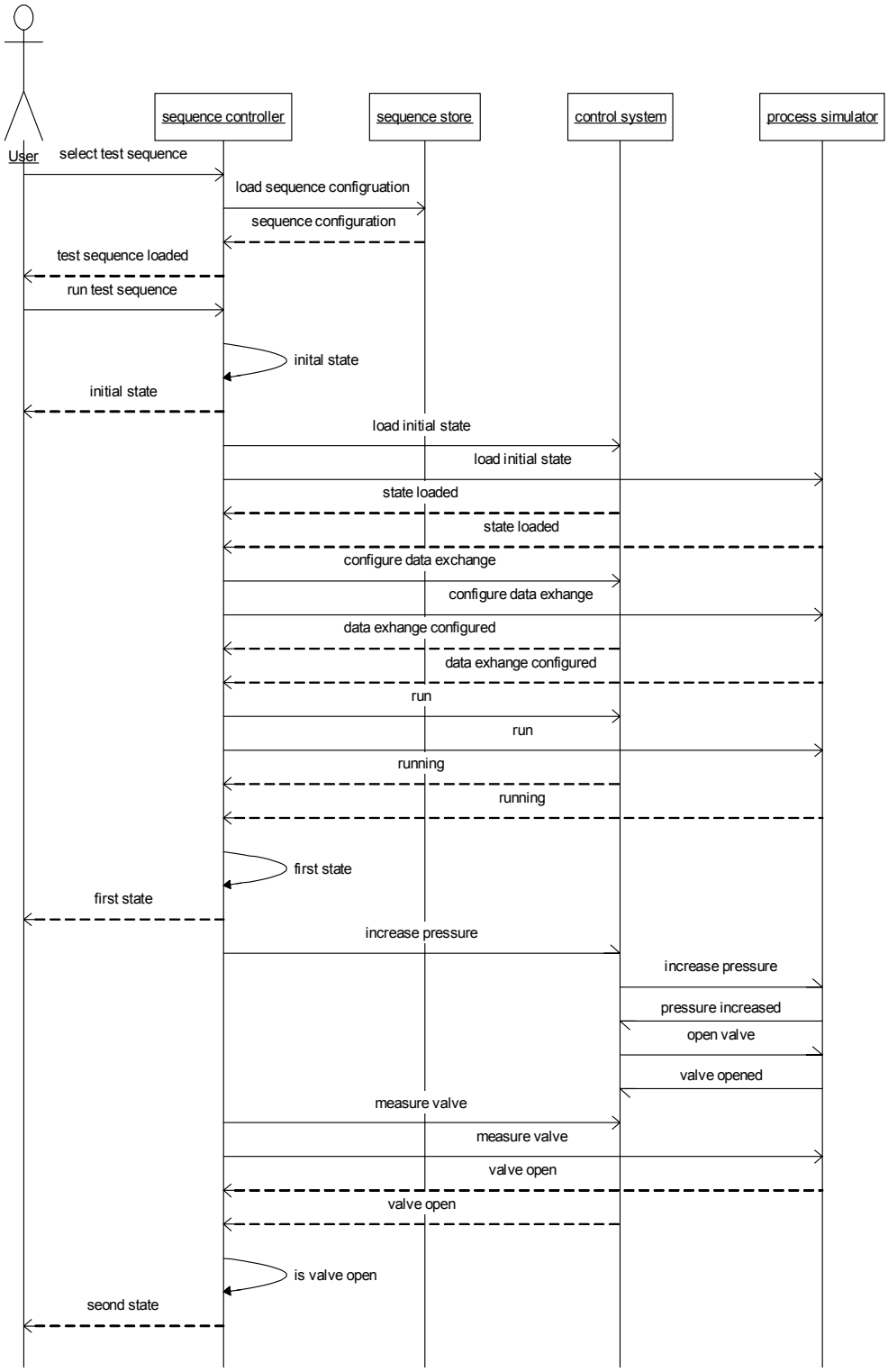


Figure 5-2. Sequence of events in the automation desing test case.

The user chooses a test sequence to run and the controller loads it from the sequence store. The user starts executing the test sequence. The test sequence is made of states and actions. The first state is the initial state, and the first action is loading the initial states to the simulator and control system. The process and control models have been configured to the simulator and control system beforehand through the configuration interface. For this the controller connects to the simulator and control system using da and extension interfaces, and orders them to load their initial states. Simulator and control system are both OPX XML data exchange servers, and controller can use the OPX XML data exchange interface to connect them to change data by OPX XML data access connections as required by the test case. When the initial states are loaded and the data exchange has been configured, the controller starts the simulator and control system as directed by the test sequence. Then the controller repeats this procedure for the following states and actions until it reaches an end state. The sequence contains verify statements (waits and if statements) which check the correctness of progress. If the sequence ends in the required end state, the test is successful.

5.2 Selecting equimpment using value-added service

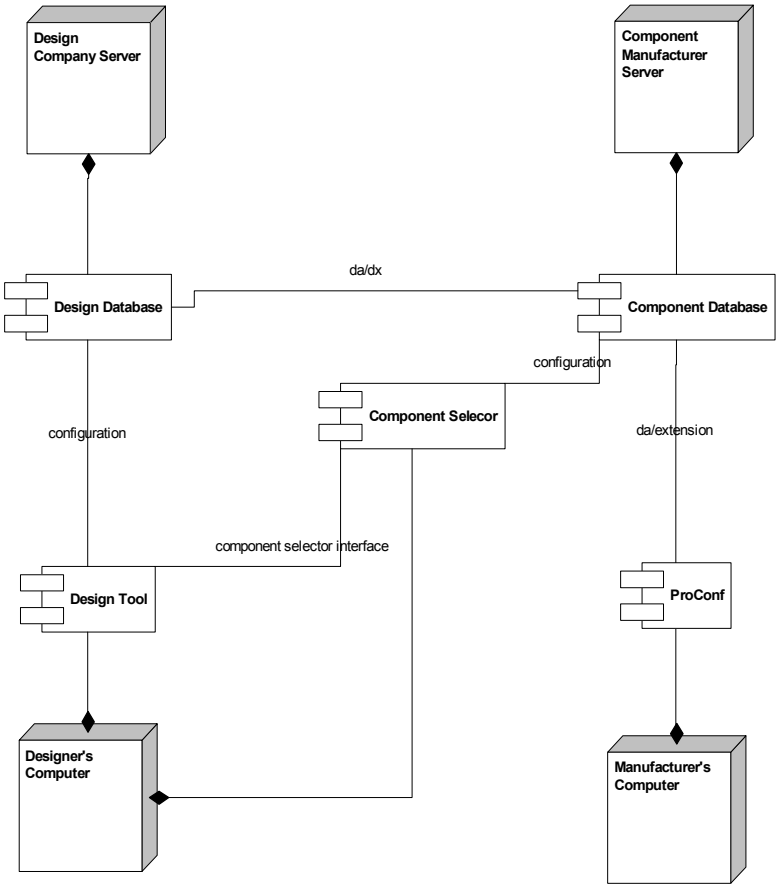


Figure 5-3. The deployment of equipment selection case.

In this case the objective is to enable the selection of equipment and use data for the equipment directly from the database of the equipment manufacturer. The physical deployment of the case is shown in *Figure 5-3*. In the case there are two core services, design database, and equipment database, each in their own node. In addition to the core services there is one value-added service, equipment selector, and two user interface components, desing tool and ProConf, the generic framework user interface tool.

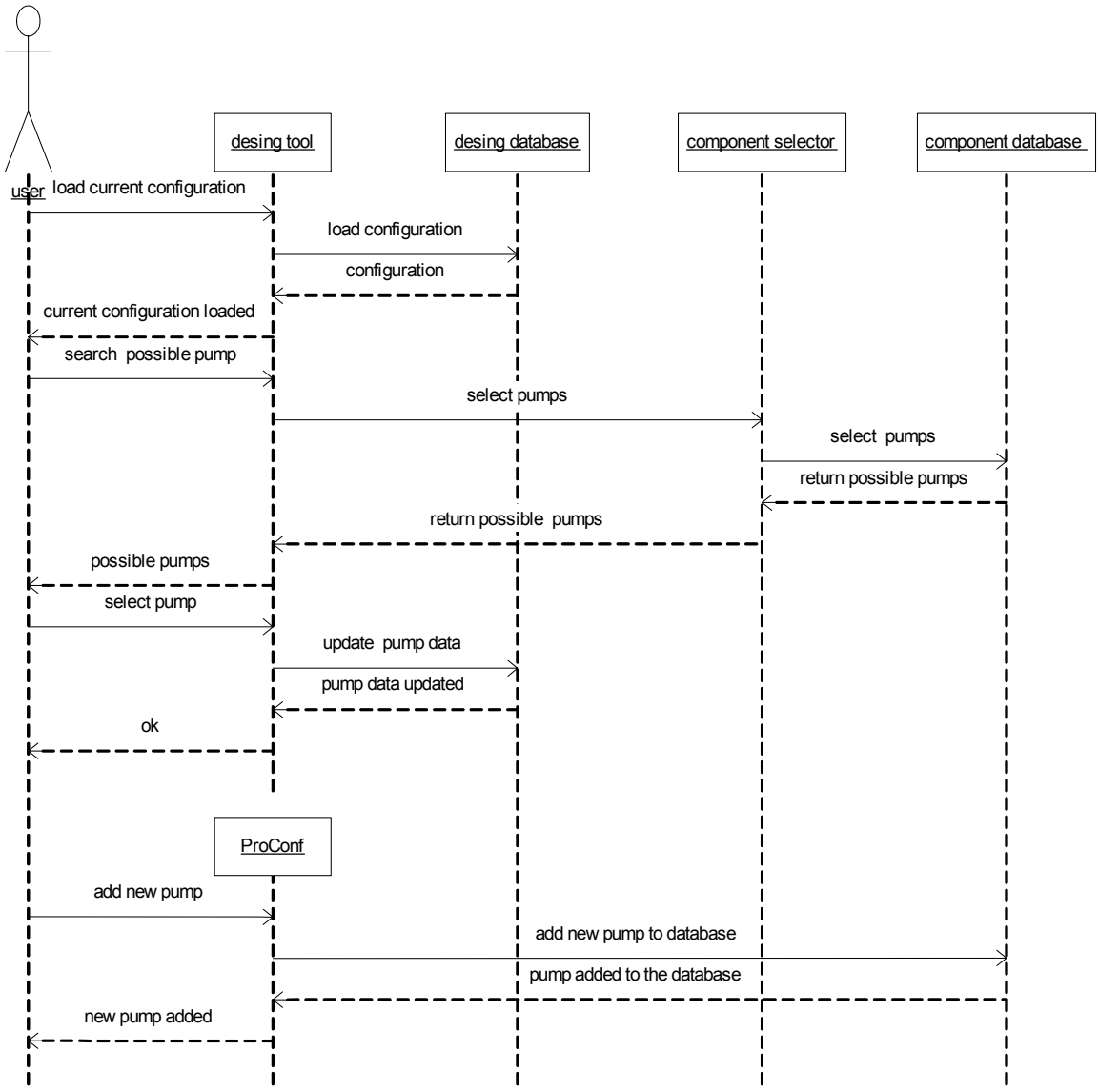


Figure 5-4. Sequence of events in the equipment selection case.

The designer needs to select a pump product to the process plant configuration. First the desiner loads the current configuration from the server of the company. Next the desiner chooses the pump from the configuration and gives selection parameters to the desing tool for searching possible pump equipments according to the criteria. The desing tool connects to the equipment selector using its specific interface, giving the selection

criteria. The selector queries the component database using the configuration interface, and selects the equipments fitting to the selection criteria and returns the selected components to the desing tool. The designer chooses the best one and inserts the new data to the design database via the design tool configuration interface to the design database. The manufacturer can keep the equipment database up to date with the framework user interface (ProConf).

6. Interface View

6.1 Introduction

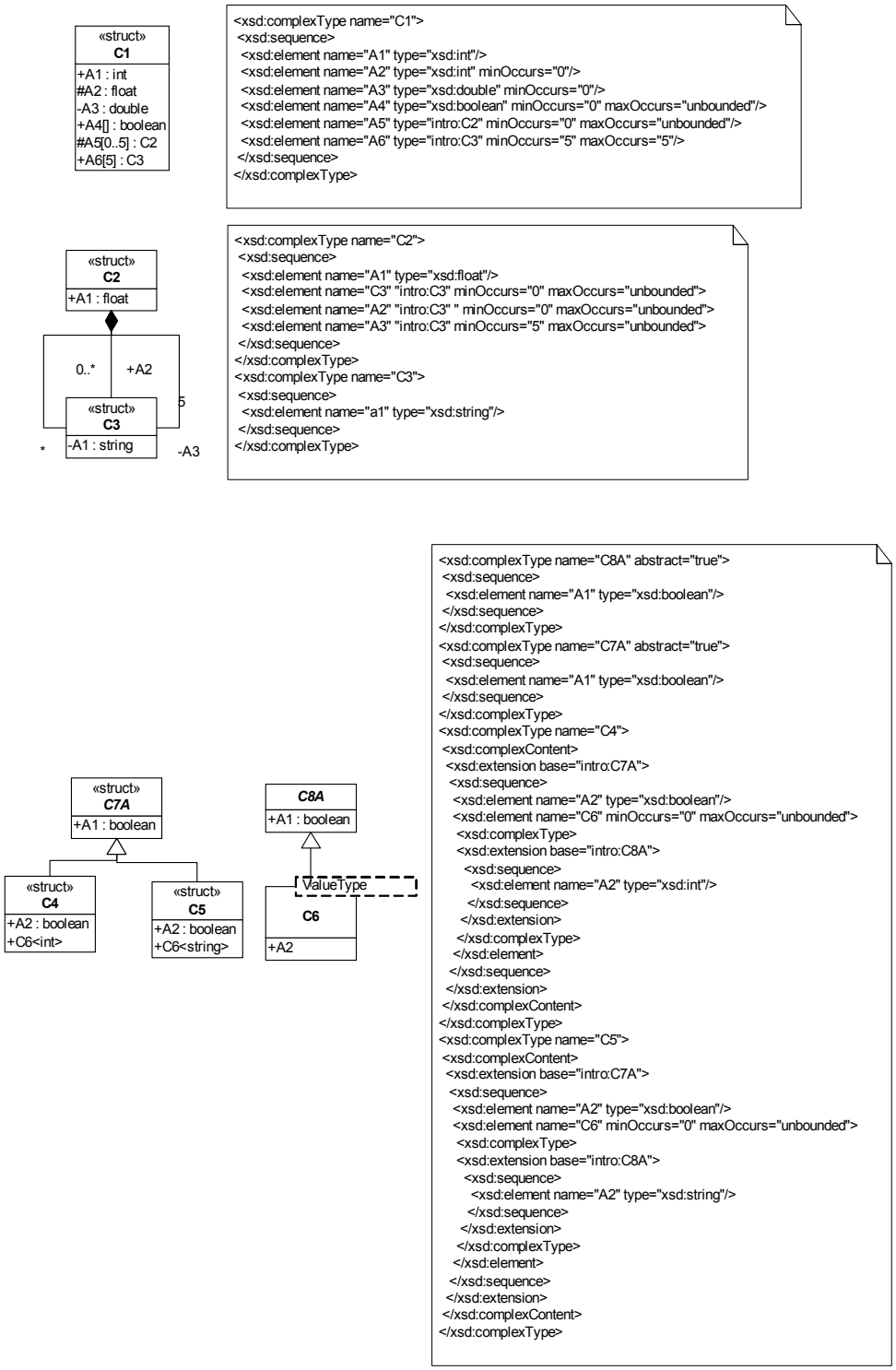


Figure 6-1. Reading instructions for the type diagrams.

The following type and operation descriptions are short, reference like descriptions of the types and operations of each interface. The normative descriptions of these interfaces are defined in the ProServ.wsdl (web service description language) document. In WSDL the types are defined by XML Schema and the in the following the Schema types are described by UML diagrams. The mappings of the UML diagrams to XML Schema are explained in *Figure 6-1*. The type descriptions describe the XML Schema named complex types. In the complex type descriptions the members table describe only the simple type schema elements. The complex type schema elements are not described because each complex type is described separately, unless it's an unnamed complex type, of course.

6.2 Framework

6.2.1 Introduction

This chapter explains the common framework concepts and types. The normative description of the types is in framework.xsd.

6.2.1.1 Namespaces

am	http://vtt.fi/webservices/ProcessSimulation/AccessManager/1.0/
cn	http://vtt.fi/webservices/ProcessSimulation/Configuration/1.0/
da	http://opcfoundation.org/webservices/XMLDA/1.0/
dx	http://opcfoundation.org/webservices/OPCDX/
g2d	http://vtt.fi/webservices/ProcessSimulation/Graphical2D/1.0/
g3d	http://vtt.fi/webservices/ProcessSimulation/Graphical3D/1.0/
fr	http://vtt.fi/webservices/ProcessSimulation/Framework/1.0/
http	http://schemas.xmlsoap.org/wsdl/http/
im	http://vtt.fi/webservices/ProcessSimulation/InstanceManager/1.0/
rm	http://vtt.fi/webservices/ProcessSimulation/RequestManager/1.0/
soap	http://schemas.xmlsoap.org/wsdl/soap/
soapenc	http://schemas.xmlsoap.org/soap/encoding/
svg	http://www.w3.org/2000/svg
wsdl	http://schemas.xmlsoap.org/wsdl/
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-Instance

6.2.1.2 Error handling

Error handling is based on SOAP exceptions. The errors are enumerated and contain an explanatory text.

6.2.2 Types

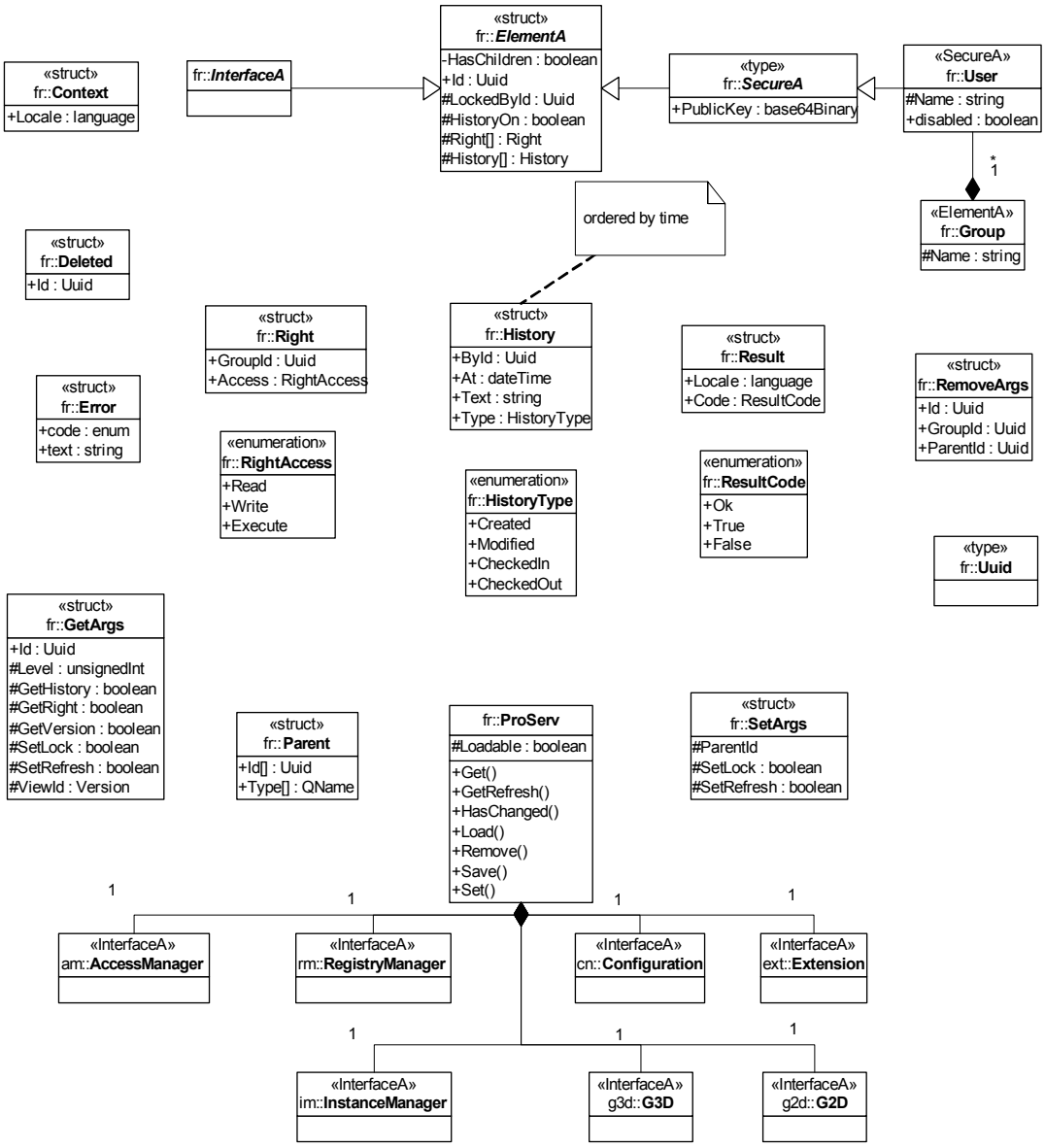


Figure 6-2. Framework types.

6.2.2.1 Context

Description

Context is the common first argument for most of the framework requests.

Members

Locale	Requested language locale. Service doesn't have to honour this.

Schema:

```
<xsd:complexType name="Context">
    <xsd:sequence>
        <xsd:element name="Locale" type="xsd:language"/>
    </xsd:sequence>
</xsd:complexType>
```

6.2.2.2 ElementA

Description

ElementA is a base type for all identifiable elements. Identifiable element can be locked. Locked elements can be read but not modified. Modification is changing element value, adding or removing a child element. The child elements derived from ElementA can be modified (unless they are locked too).

Members

HasChildren	True if the element has child elements with maxOccurs greater than one and which are derived from the ElementA. This is needed for gets, which return only the asked number of child element levels.
Id	Uniquely identifies the element.
LockedById	Uuid of the user who has the element locked.
HistoryOn	If true then operations on the element are recorded. See History for more information.

Schema

```
<xsd:complexType name="ElementA" abstract="true">
    <xsd:sequence>
        <xsd:element name="HasChildren" type="xsd:boolean" default="false" minOccurs="0"/>
        <xsd:element name="Id" type="fr:Uuid"/>
        <xsd:element name="LockedById" type="fr:Uuid" minOccurs="0"/>
        <xsd:element name="HistoryOn" type="xsd:boolean" default="false" minOccurs="0"/>
        <xsd:element name="VersionOn" type="xsd:boolean" default="false" minOccurs="0"/>
        <xsd:element name="Right" type="fr:Right" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="History" type="fr:History" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="Version" type="fr:Version" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

6.2.2.3 ElementIdentification

Description

Uniquely identifies an element.

Members

Id	Identifies the deleted element.

Schema

```
<xsd:complexType name="ElementIdentification">
    <xsd:sequence>
        <xsd:element name="Id" type="fr:Uuid"/>
    </xsd:sequence>
</xsd:complexType>
```

6.2.2.4 Error

Description

Error describes data that is notified as soap faults.

Members

Code	Error code.
Text	Explanatory text of the error in the given language (see FR.Result).

ErrorCode

E_EXIST	Element or some other entity does not exist.
E_FAIL	Operation failed partly or completely.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_INSUFFICIENT_RIGHTS	Operation could not be completed because of missing access rights. Note that malicious user could get compromising data by this.
E_INSUFFICIENT_RESOURCES	Not enough resources to complete the request.
E_LOCKED	Operation could no be completed beacuse element(s) were locked.
E_NOT_IMPLEMENTED	The operation is not implemented.
E_NOT_LOADABLE	The element is not loadable.
E_REQUEST	Request did not conform to the wsdl.

Schema

```
<xsd:complexType name="Error">
    <xsd:attribute name="Code" type="fr:ErrorCode"/>
    <xsd:attribute name="Text" type="xsd:string"/>
</xsd:complexType>
<xsd:simpleType name="ErrorCode">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="E_EXIST"/>
        <xsd:enumeration value="E_FAIL"/>
        <xsd:enumeration value="E_ILLEGAL_ARGUMENT"/>
        <xsd:enumeration value="E_INSUFFICIENT_RIGHTS"/>
        <xsd:enumeration value="E_INSUFFICIENT_RESOURCES"/>
        <xsd:enumeration value="E_LOCKED"/>
        <xsd:enumeration value="E_NOT_IMPLEMENTED"/>
        <xsd:enumeration value="E_NOT_LOADABLE"/>
        <xsd:enumeration value="E_REQUEST"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.2.2.5 GetArgs

Description

GetArgs describes arguments for the get operation.

Members

Id	Identifies the top element to get.
Level	How many levels to get. Zero means getting everything, one means getting only the simple type elements and complex type elements not derived from Element A. Two means that get with level one if applied to complex type child elements derived from ElementA, and so on. Default is 1.
GetHistory	If true return History elements, default true.
GetRight	If true return Rights elements, default true.
SetLock	If true each returned lockable element is locked. Default is false.
SetRefresh	If true each returned element is marked to be returned with GetRefresh.

Schema

```
<xsd:complexType name="GetArgs">
    <xsd:sequence>
        <xsd:element name="Id" type="fr:Uuid" minOccurs="0"/>
        <xsd:element name="Level" type="xsd:unsignedInt" minOccurs="0"/>
        <xsd:element name="GetHistory" type="xsd:boolean" default="true" minOccurs="0"/>
        <xsd:element name="GetRight" type="xsd:boolean" default=" true " minOccurs="0"/>
        <xsd:element name="SetLock" type="xsd:boolean" default="false" minOccurs="0"/>
        <xsd:element name="SetRefresh" type="xsd:boolean" default="false" minOccurs="0"/>
        <xsd:element name="ViewId" type="fr:Uuid" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

6.2.2.6 Group

Description

Group is a set of users. Each identifiable element has rights, and each right element grants access rights for the users it refers. It can refer either to and individual user, or to a group of users. Groups can not be removed, only disabled. When group is removed with remove request, it will only be disabled.

Members

Name	Friendly name for the user.
Disabled	If exists and true the group has been disabled.
User	The user who belongs to this group.

Schema

```
<xsd:complexType name="Group">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Disabled" type="xsd:boolean"  minOccurs="0" default="false"/>
                <xsd:element ref="fr:User" minOccurs="0" maxOccurs="unbounded"/>
                <!--<xsd:element name="User" type="fr:User" minOccurs="0" maxOccurs="unbounded"/>-->
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.2.2.7 History

Description

When the HistoryOn attribute of parent is true then keeps a log of parent modifications, i.e. changes of values, adding and removing of child elements. The log entries are added automatically by the server.

Members

At	When the log was made. Added automatically by the server. Readonly.
ById	The Uuid of the user who made the modification. Added automatically by the server. Readonly.
Type	Type of the logged event. Readonly.
Description	Prose. Language specific.

HistoryType

Created	The element was created.
Modified	The element was modified.
CheckedIn	The element was checked in. (Where is the version number?)
ChekckedOut	The element was checked out. (Where is the version number?)

Schema

```
<xsd:complexType name="History">
    <xsd:sequence>
        <xsd:element name="ById" type="fr:Uuid"/>
        <xsd:element name="At" type="xsd:dateTime"/>
        <xsd:element name="Text" type="xsd:string"/>
        <xsd:element name="Type" type="fr:HistoryType"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="HistoryType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Created"/>
        <xsd:enumeration value="Modified"/>
        <xsd:enumeration value="CheckedIn"/>
        <xsd:enumeration value="CheckedOut"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.2.2.8 InterfaceA

Description

InterfaceA is a base class for interfaces.

Members

Schema

```
<xsd:complexType name="InterfaceA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.2.2.9 Parent

Description

Parent identifies the parent element of the element requested by get.

Members

Id	Identifies the parent.
ElementType	Type of the parent.

Schema

```
<xsd:complexType name="Parent">
    <xsd:sequence>
        <xsd:element name="Id" type="fr:Uuid"/>
        <xsd:element name="Type" type="xsd:QName"/>
    </xsd:sequence>
</xsd:complexType>
```

6.2.2.10 ProServ

Description

ProServ is the root element for the interface elements, and the element that get operation without an id will return.

Members

Loadable	If true then the service is loadable. This means that the requests HasChanged, Load, and Save are implemented.

Schema

```
<xsd:complexType name="ProServ">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Loadable" type="xsd:boolean" default="false" minOccurs="0"/>
                <xsd:element ref="am:AccessManager"/>
                <xsd:element ref="im:InstanceManager"/>
                <xsd:element ref="rm:RegistryManager"/>
                <xsd:element ref="cn:Configuration"/>
                <xsd:element ref="ext:Extension"/>
                <xsd:element ref="g2d:G2D"/>
                <xsd:element ref="g3d:G3D"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.2.2.11 RemoveArgs

Description

RemoveArgs identifies the element to remove. Used by remove operation.

Members

Id	Uuid of the element to remove. Mutually exclusive with ParentId and GroupId.
ParentId	Identifies the parent of a group to remove. Mutually exclusive with Id.
GroupId	Identifies the group to remove. Mutually exclusive with Id.

Schema

```
<xsd:complexType name="RemoveArgs">
    <xsd:sequence>
        <xsd:element name="Id" type="fr:Uuid" minOccurs="0"/>
        <xsd:element name="ParentId" type="fr:Uuid" minOccurs="0"/>
        <xsd:element name="GroupId" type="fr:Uuid" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

6.2.2.12 Result

Description

Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Members

Locale	Used language locale.

ResultCode

Ok	Everything as expected.
True	Boolean operation returning true.
False	Boolean operation returning false.

Schema

```
<xsd:complexType name="Result">
    <xsd:sequence>
        <xsd:element name="Code" type="fr:ResultCode"/>
        <xsd:element name="Locale" type="xsd:language"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="ResultCode">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Ok"/>
        <xsd:enumeration value="True"/>
        <xsd:enumeration value="False"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.2.2.13 Right

Description

Right defines the access rights for the containing element. To access an element you must have rights to the element. The superuser has full rights to the element; others have the rights defined by the rights elements.

Members

Access	The access rights for the users of the given group.
GroupId	Identifies the group this rights definition concerns. The group is defined in this service's access domain (managed by the access manager).

RightAccess

Read	Read rights, means you can get the element.
Write	Write rights, means you can get and set the child element values and add and remove child elements.
Execute	Execute rights, means you can execute the element if it is executable (see Extension). Execute rights automatically grants read rights.

Schema

```
<xsd:complexType name="Right">
    <xsd:sequence>
        <xsd:element name="Access" type="fr:RightAccess"/>
        <xsd:element name="GroupId" type="fr:Uuid"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="RightAccess">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Read"/>
        <xsd:enumeration value="Write"/>
        <xsd:enumeration value="Execute"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.2.2.14 SetArgs

Description

SetArgs describe arguments to set.

Members

ParentId	If adding a new element then the id of the parent element must be given.
SetLock	If true every given element is locked.
SetRefresh	If true every given element is marked to be returned with refresh.

Schema

```
<xsd:complexType name="SetArgs">
    <xsd:sequence>
        <xsd:element name="ParentId" type="fr:Uuid"/>
        <xsd:element name="SetLock" type="xsd:boolean" default="false" minOccurs="0"/>
        <xsd:element name="SetRefresh" type="xsd:boolean" default="false" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

6.2.2.15 User

Description

User is the smallest authentication unit. Each request has a user, and is granted or denied based on the rights the user in question has. The user authentication is done by the service based on the user's credentials. Users can not be removed, only disabled. When user is removed with remove request, it will only be disabled.

Members

Name	Friendly name for the user.
Disabled	If true the user has been disabled.

Schema

```
<xsd:complexType name="User">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string"  minOccurs="0"/>
                <xsd:element name="Disabled" type="xsd:boolean" default="false"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.2.2.16 Uuid

Description

Uuid is a 36 byte value using hexadecimal digits in the following format: `12345678-1234-1234-1234-123456789ABC`. It is used to indetify most framework entities. Each new Uuid must be unique. The algortihm of Uuid generation is described in (DCE 1998). Null is 00000000-0000-0000-0000-000000000000.

Schema

```
<xsd:simpleType name="Uuid">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9A-F]{8}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]{12}"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.2.3 Operations

6.2.3.1 Get

Description
Get identifiable element by id.

6.2.3.1.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
GetArgs	GetArgs describes arguments for the get operation.

Schema

```
<xsd:element name="GetRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="GetArgs" type="fr:GetArgs" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.1.2 Response

Arguments

Result	See Result.
Parent	See Parent.
ElementA	See ElementA.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_EXIST	Element or some other entity does not exist.

Schema

```
<xsd:element name="GetResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
            <xsd:element name="Parent" type="fr:Parent" minOccurs="0"/>
            <xsd:element name="ElementA" type="fr:ElementA"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.2 GetRefresh

Description
Get refresh elements. All identifiable elements marked with get or set from the same session to be in the refresh mechanism are monitored for modifications. GetRefresh request returns the modified and deleted identifiable elements since the last GetRefresh request. A session is one HTTP connection. If the HTTP connection is broken all refresh information is lost.

6.2.3.2.1 Request

Arguments

Context	See Context.

Schema

```
<xsd:element name="GetRefreshRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.2.2 Response

Arguments

Result	See Result.
ElementA	See ElementA.
Deleted	Identifies a deleted element.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_EXIST	Element or some other entity does not exist.

Schema

```
<xsd:element name="GetRefreshResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
            <xsd:element name="ElementA" type="fr:ElementA" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Deleted" type="fr:ElementIdentification" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.3 HasChanged

Description

True if service content has changed since last load. The operation is implemented only if the service is loadable.

6.2.3.3.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.

Example

```
<xsd:element name="HasChangedRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.3.2 Response

Arguments

Result	Code is true if the content has changed.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_NOT_IMPLEMENTED	The operation is not implemented.

Example

```
< <xsd:element name="HasChangedResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.4 Load

Description

Load the service content from disk to memory; all changes since last save are lost. The operation is implemented only if the service is loadable.

6.2.3.4.1 Request

Arguments

Context	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Schema

```
<xsd:element name="LoadRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
        </xsd:sequence>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.4.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_NOT_IMPLEMENTED	The operation is not implemented.

Schema

```
<xsd:element name="LoadResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.5 Remove

Description

Remove the given identifiable element. The given element and its child elements are removed.

6.2.3.5.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
RemoveArgs	Identifies the element(s) to remove.

Schema

```
<xsd:element name="RemoveRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="RemoveArgs" type="fr:RemoveArgs"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.5.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_EXIST	Element or some other entity does not exist.

Schema

```
<xsd:element name="RemoveResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.6 Save

Description

Save the configuration to disk. The operation is implemented only if the service is loadable.

6.2.3.6.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.

Schema

```
<xsd:element name="SaveRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
        </xsd:sequence>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.6.2 Response

Arguments

Result	

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_NOT_IMPLEMENTED	The operation is not implemented.

Schema

```
<xsd:element name="SaveResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.7 Set

Description

Create or modify identifiable element. If you are adding a new element the parent id must be given.

6.2.3.7.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
SetArgs	See SetArgs.
ElementA	See ElementA.

Schema

```
<xsd:element name="SetRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="SetArgs" type="fr:SetArgs"/>
            <xsd:element name="ElementA" type="fr:ElementA"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.2.3.7.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_EXIST	Element or some other entity does not exist.

Schema

```
<xsd:element name="SetResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.3 Access Manager

6.3.1 Introduction

This chapter explains the access manager interface types and operations. Access manager is an interface providing common security model for one or more services. The normative description of the interface types is in access_manager.xsd.

6.3.2 Types

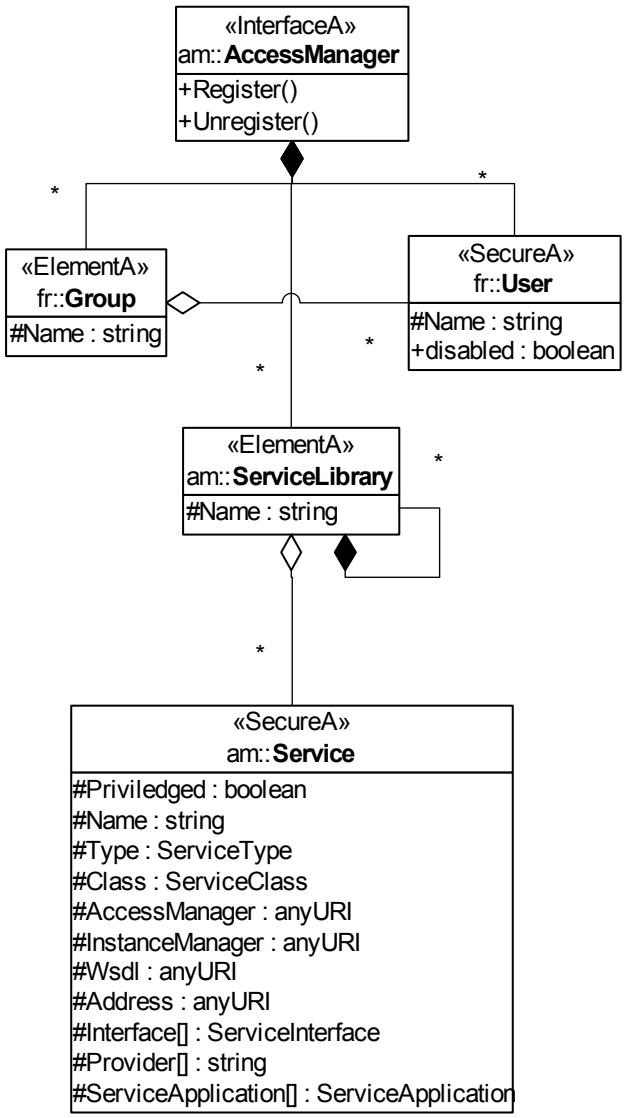


Figure 6-3. Types of access manager.

6.3.2.1 AccessManager

Description

AccessManager is the type representing the corresponding interface.

Members

Group	The groups of the access manager.
User	All the users in all the groups.

Schema

```
<xsd:complexType name="AccessManager">
    <xsd:complexContent>
        <xsd:extension base="fr:InterfaceA">
            <xsd:sequence>
                <xsd:element ref="fr:Group" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="fr:User" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.3.2.2 Service

Description

Service reprecents class of service applications that share the same wsdl. Service is a collection of interfaces and represents service(s) offered to the client. A service must have one, and can have all the interfaces described in this document. Access manager is interested only in the id of the service, its public key and the priviledged value. The other members are for instance and registry managers.

Members

Priviledged	If exists and true then the service is allowed to use the register and unregister callbacks.
Name	Friendly name for the service.
Type	Type of the service.
Class	Class of the service. Groups services by functionality.
AccessManager	URI of the access manager of the service, the security service provider for this service.
InstanceManager	URI of the instance manager of the service. If exists then the service is a dynamic service and can be managed by the instance manager interface.
Wsdl	URI of the HTTP server were the WSDL of the service can be fetch with http get.
Address	URI of the service. If exists then the URI of the service is always the same, and there can only be one service instance running at any given time. This is always the case if the service does not have an instance manager.

Interface	List of interfaces the service supports.
Provider	List of strings defining the providers of the service.
ServiceApplication	List of applications conforming to the service wsdl. Each application has a different implementation.

ServiceType

ValueAdded	The WSDL is free.
Framework	The WSDL conforms to this specification. Framework service interfaces are all static i.e. being predefined. The client does not have to ask framework service interface descriptions because they must always be as described in this document. All the client needs to know is the supported interfaces. This is seen by interfaces the the service returns. The ids of the interfaces are fixed. Framework service must implement at least on core service interface and at least one utility service interface.
Core	Same as framework, except does not implement any utility interface.
Utility	Same as framework, except does not implement any core interface, and implements more than one utility interface.
AccessManager	Implements only access manager interface.
InstanceManager	Implements only instance manager interface.
RegistryManager	Implements only registry manager interface.

ServiceClass

Unknown	None of the below.
ControlSystem	Supports data access interface.
Database	Supports configuration interface.
Simulator	Supports configuration, extension, and data access.

ServiceInterface

AccessManager	Supports the interface described in the access manager chapter.
RegistryManager	Supports the interface described in the registry manager chapter.
InstanceManager	Supports the interface described in the instance manager chapter.
Configuration	Supports the interface described in the configuration chapter.
Extension	Supports the interface described in the extension chapter.
Graphic2D	Supports the interface described in the graphic 2d chapter.
Graphic3D	Supports the interface described in the graphic 3d chapter.
DataAccess	Supports the OPC XML data access interface.
FastDataAccess	Supports the interface described in the fast data access chapter.
DataExchange	Supports the OPC XML data exchange interface.

Schema

```
<xsd:complexType name="Service">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Priviledged" type="xsd:boolean" minOccurs="0" default="false"/>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Type" type="am:ServiceType" minOccurs="0"/>
                <xsd:element name="Class" type="am:ServiceClass" minOccurs="0"/>
```

```
                <xsd:element name="AccessManager" type="xsd:anyURI" minOccurs="0"/>
                <xsd:element name="InstanceManager" type="xsd:anyURI" minOccurs="0"/>
                <xsd:element name="Wsdl" type="xsd:anyURI" minOccurs="0"/>
                <xsd:element name="Address" type="xsd:anyURI" minOccurs="0"/>
                <xsd:element name="Interface" type="am:ServiceInterface" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="Provider" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="ServiceApplication" type="am:ServiceApplication" minOccurs="0"
    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="ServiceClass">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Unknown"/>
        <xsd:enumeration value="ControlSystem"/>
        <xsd:enumeration value="Database"/>
        <xsd:enumeration value="Simulator"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ServiceInterface">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="AccessManager"/>
        <xsd:enumeration value="RegistryManager"/>
        <xsd:enumeration value="InstanceManager"/>
        <xsd:enumeration value="Configuration"/>
        <xsd:enumeration value="Extension"/>
        <xsd:enumeration value="Graphic2D"/>
        <xsd:enumeration value="Graphic3D"/>
        <xsd:enumeration value="DataAccess"/>
        <xsd:enumeration value="FastDataAccess"/>
        <xsd:enumeration value="DataExchange"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ServiceType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ValueAdded"/>
        <xsd:enumeration value="Framework"/>
        <xsd:enumeration value="Core"/>
        <xsd:enumeration value="Utillity"/>
        <xsd:enumeration value="AccessManager"/>
        <xsd:enumeration value="InstanceManager"/>
        <xsd:enumeration value="RegistryManager"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.3.2.3 ServiceApplication

Description

Service application represents an implementation of service. Service application divides services into set of services which have common parameters, mainly the same module.

Members

Module	Path to the implementation module.

Schema

```
<xsd:complexType name="ServiceApplication">
    <xsd:sequence>
        <xsd:element name="Module" type="xsd:string"/>
        <xsd:element name="ServiceInstance" type="am:ServiceInstance" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

6.3.2.4 ServiceInstance

Description

Service instance represents an instance of service application. The user who owns the instance process is configured and is not shown to clients.

Members

Address	URI of the service. If the service is static then the address is given be the Address element of Service, and there can only be one instance running at any given time. If the service can not be started and stopped via instance manager the application information is not stored and handled at all.
Directory	The working directory of the instance.
Running	True if the instance is running.
Argument	Arguments for the instance.

Schema

```
<xsd:complexType name="ServiceInstance">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Address" type="xsd:anyURI" minOccurs="0"/>
                <xsd:element name="Directory" type="xsd:string"/>
                <xsd:element name="Running" type="xsd:boolean"/>
                <xsd:element name="Argument" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.3.2.5 ServiceLibrary

Description

Service library is a container for services.

Members

Name	Friendly name for the library.

Schema

```
<xsd:complexType name="ServiceLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Service" type="am:Service" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="ServiceLibrary" type="am:ServiceLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.3.3 Operations

6.3.3.1 AccessManagerNotify

Description

Callback for notificating changes in the users and groups. The registering service must implement this to be notified of changes in users and groups. The callback will be called whenever a user or group data changes. The first callback call will contain all groups and users, the rest only the changed group or user.

6.3.3.1.1 Request

Arguments

Group	The group that has changed.
User	The user that has changed.

Schema

```
<xsd:element name="AccessManagerNotifyRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Group" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="fr:User" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.3.3.1.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_NOT_IMPLEMENTED	The operation is not implemented.

Schema

```
<xsd:element name="AccessManagerNotifyResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.3.3.2 Register

Description

Subscribe notifications of changes in the users and groups. Only the priviledged services configured to the access manager can use the access control functions i.e. register and unregister the callbacks. The service wanting to register a callback must implement AccessManagerNotify callback (see above).

6.3.3.2.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
Callback	URI for the notify callback. The uri must implement the AccessManagerNotify callback (see above).

Schema

```
<xsd:element name="RegisterRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context"/>
            <xsd:element name="Callback" type="xsd:anyURI"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.3.3.2.2 Response

Arguments

Result	Context is the common first argument for most of the framework requests.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_NOT_IMPLEMENTED	The operation is not implemented.

Schema

```
<xsd:element name="RegisterResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.3.3.3 Unregister

Description

Unsubscribe the previously registered callback.

6.3.3.3.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
Callback	The same uri that was used to register the callback. The callbacks are registered by the callback uri, and only one callback to any given uri is allowed.

Schema

```
<xsd:element name="UnregisterRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Context"/>
            <xsd:element name="Callback" type="xsd:anyURI"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.3.3.3.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes. AccessManager uses this to verify that the client is still alive.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_EXIST	Element or some other entity does not exist.
E_NOT_IMPLEMENTED	The operation is not implemented.

Schema

```
<xsd:element name="UnregisterResponse">
<xsd:complexType>
    <xsd:sequence>
        <xsd:element ref="fr:Result"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

6.4 Registry Manager

6.4.1 Introduction

This chapter explains the Registry Manager interface. The normative description of the types is in registry_manager.xsd.

6.4.2 Types

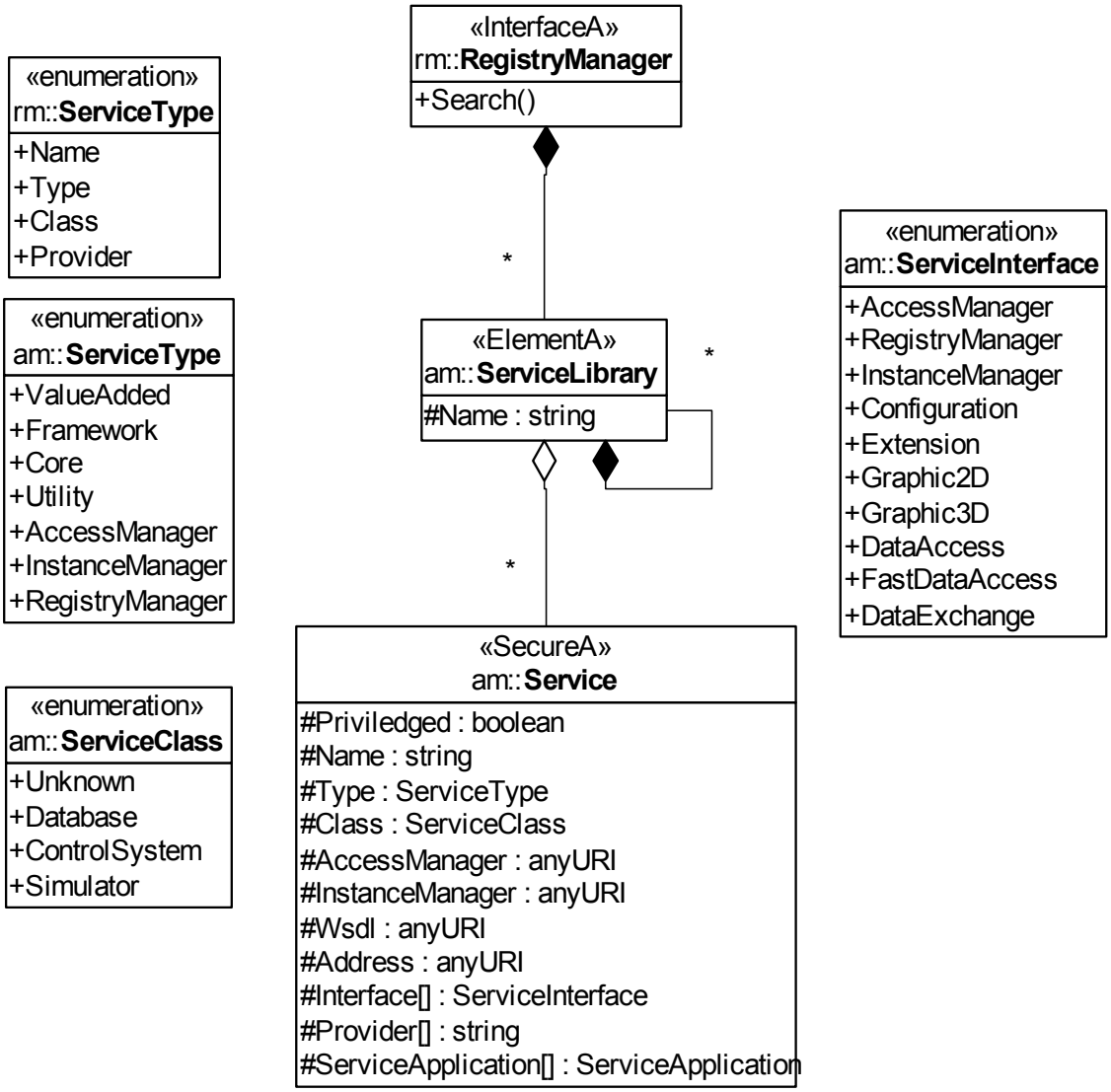


Figure 6-4. Types of registry manager.

6.4.2.1 RegistryManager

Description

RegistryManager is the type representing the corresponding interface.

Members

Schema:

```
<xsd:complexType name="RegistryManager">
        <xsd:sequence>
            <xsd:element ref="am:ServiceLibrary" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:complexType>
```

6.4.3 Operations

6.4.3.1 Search

Description

Search services by given criteria.

6.4.3.1.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
Key	The search key. Defines a value to compare to value in the service. Depending on the type argument is the service name, type or class, or provider.
SearchType	Defines what the key value represents.

SearchType

Name	Search is by name of the service. If null all services are returned.
Type	Search is by type of the service.
Class	Search is by class of the service.
Provider	Search is by provider.

Schema

```
<xsd:element name="SearchRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context"/>
            <xsd:element name="Key" type="xsd:string"/>
            <xsd:element name="SearchType" type="rm:SearchType"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:simpleType name="SearchType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Name"/>
        <xsd:enumeration value="Type"/>
        <xsd:enumeration value="Class"/>
        <xsd:enumeration value="Provider"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.4.3.1.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.
Service	Services with matching values.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="SearchResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
            <xsd:element name="Service" type="am:Service" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.5 Instance Manager

6.5.1 Introduction

This chapter explains the instance manager interface. Instance manager provides services for launching dynamic services. The normative description of the types is in InstanceManager.xsd.

6.5.2 Types

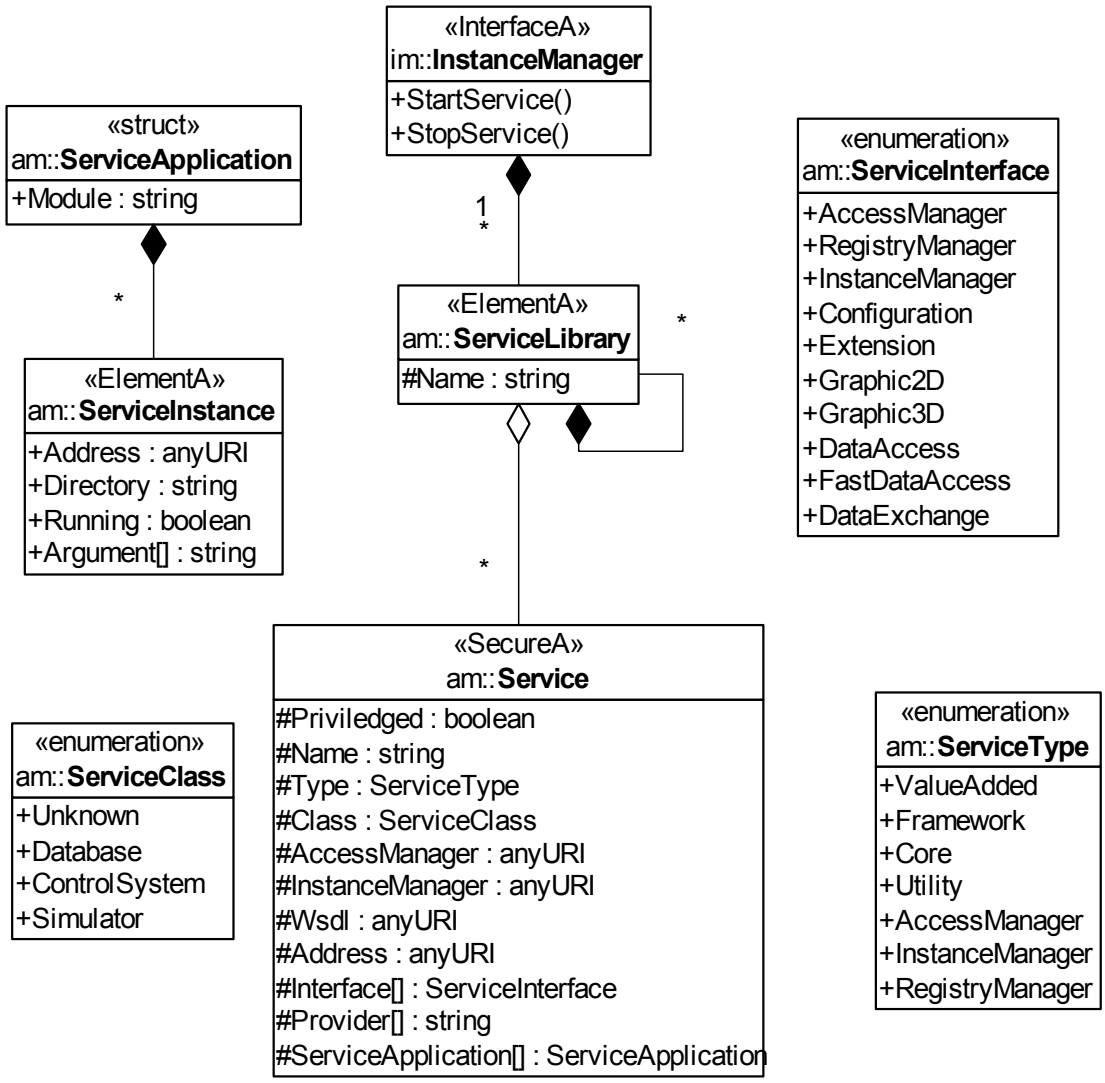


Figure 6-5. Types of instance manager.

6.5.2.1 InstanceManager

Description

InstanceManager is the type representing the corresponding interface.

Members

Schema

```
<xsd:complexType name="InstanceManager">
    <xsd:complexContent>
        <xsd:extension base="fr:InterfaceA">
            <xsd:sequence>
                <xsd:element ref="am:ServiceLibrary" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.5.3 Operations

6.5.3.1 StartService

Description

StartService is operation for starting the given service.

6.5.3.1.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
InstanceId	Identifies the instance to start.

Schema

```
<xsd:element name="StartServiceRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context"/>
            <xsd:element name="InstanceId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.5.3.1.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="StartServiceResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.5.3.2 StopService

Description

StopService operation is for stopping the given service. This is only a request to stop, if the service does not stop because it has open client connections, it is not violently stopped.

6.5.3.2.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
InstanceId	Identifies the instance to stop.

Schema

```
<xsd:element name="StopServiceRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context"/>
            <xsd:element name="InstanceId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.5.3.2.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes. Returns false if the service does not agree to stop.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="StopServiceResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Result" type="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.6 Configuration

6.6.1 Introduction

This chapter explains the configuration interface. The normative description of the types is in configuration.xsd.

6.6.2 Types

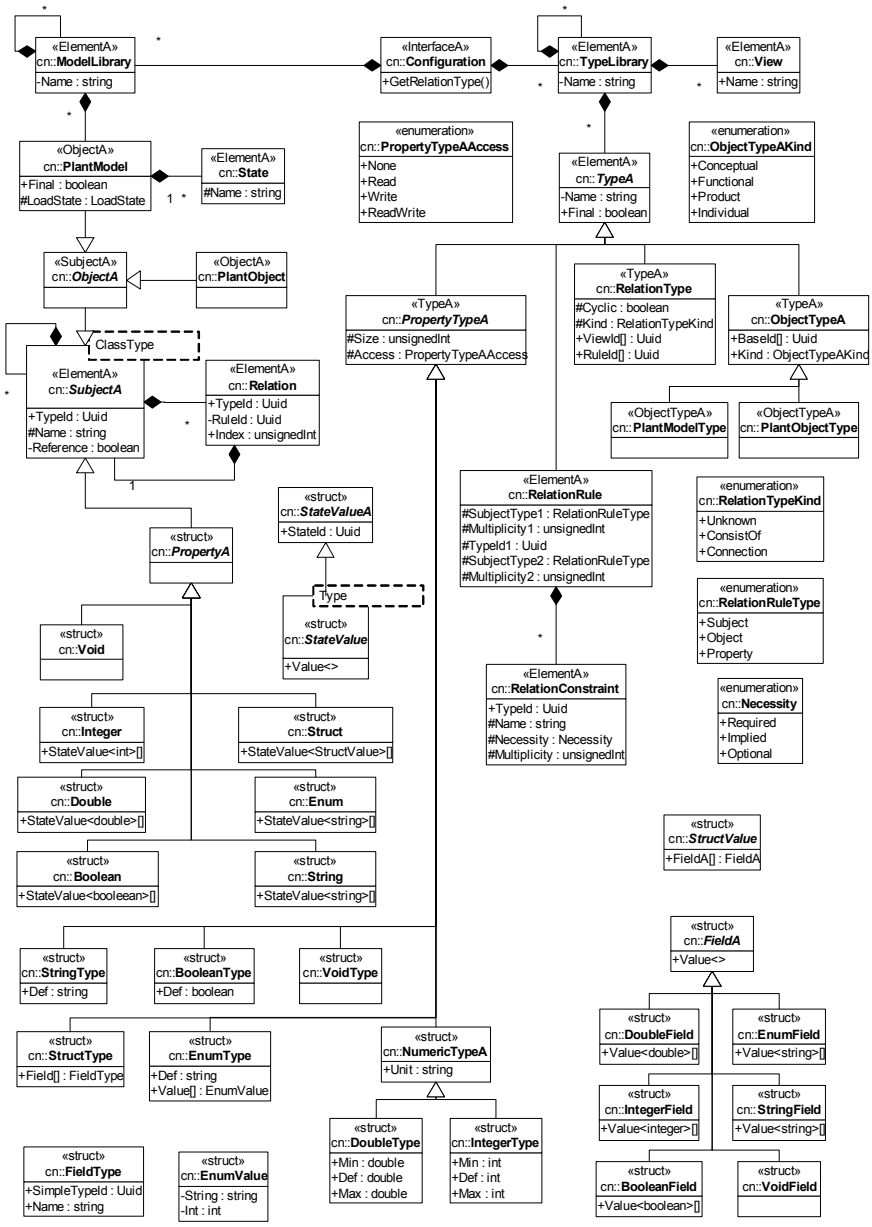


Figure 6-6. Types of configuration interface.

6.6.2.1 Boolean

Description

Boolean is a representation of logical values.

Members

Value	The double values.

Schema

```
<xsd:complexType name="Boolean">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyA">
            <xsd:sequence>
                <xsd:element name="StateValue" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:complexContent>
                            <xsd:extension base="cn:StateValueA">
                                <xsd:sequence>
                                    <xsd:element name="Value" type="xsd:boolean" maxOccurs="unbounded"/>
                                </xsd:sequence>
                            </xsd:extension>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.2 BooleanField

Description

Boolean field is a type for Boolean field values.

Members

Value	Defines the field values.

Schema

```
<xsd:complexType name="BooleanField">
    <xsd:complexContent>
        <xsd:extension base="cn:FieldA">
            <xsd:sequence>
                <xsd:element name="Value" type="xsd:boolean" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.3 BooleanType

Description

Boolean type is a class for Boolean value instances. Boolean type defines common properties for all Boolean instances from this class.

Members

Min	Defines the allowed minimum value for the instance values.
Def	Defines the default value for the instance values. Used when the value is created without specifying initial value.
Max	Defines the alloved maximum value for the instance values.

Schema

```
<xsd:complexType name="DoubleType">
    <xsd:complexContent>
        <xsd:extension base="cn:NumericTypeA">
            <xsd:sequence>
                <xsd:element name="Min" type="xsd:double" minOccurs="0"/>
                <xsd:element name="Def" type="xsd:double" minOccurs="0"/>
                <xsd:element name="Max" type="xsd:double" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.4 Configuration

Description

Configuration is the type representing the corresponding interface.

Members

Schema

```
<xsd:complexType name="Configuration">
    <xsd:complexContent>
        <xsd:extension base="fr:InterfaceA">
            <xsd:sequence>
                <xsd:element name="TypeLibrary" type="cn:TypeLibrary" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="ModelLibrary" type="cn:ModelLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.5 Double

Description

Double is a representation of double precision (64 bit) floating point numbers as specified by the IEEE standard for floating point arithmetic.

Members

Value	The double values.

Schema

```
<xsd:complexType name="Double">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyA">
            <xsd:sequence>
                <xsd:element name="StateValue" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:complexContent>
                            <xsd:extension base="cn:StateValueA">
                                <xsd:sequence>
                                    <xsd:element name="Value" type="xsd:double" maxOccurs="unbounded"/>
                                </xsd:sequence>
                            </xsd:extension>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.6 DoubleField

Description

Double field is a type for double field values.

Members

Value	Defines the field values.

Schema

```
<xsd:complexType name="DoubleField">
    <xsd:complexContent>
        <xsd:extension base="cn:FieldA">
            <xsd:sequence>
                <xsd:element name="Value" type="xsd:double" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.7 DoubleType

Description

Double type is a class for double value instances. Double type defines common properties for all double instances from this class.

Members

Min	Defines the allowed minimum value for the instance values.
Def	Defines the default value for the instance values. Used when the value is created without specifying initial value.
Max	Defines the alloved maximum value for the instance values.

Schema

```
<xsd:complexType name="DoubleType">
    <xsd:complexContent>
        <xsd:extension base="cn:NumericTypeA">
            <xsd:sequence>
                <xsd:element name="Min" type="xsd:double" minOccurs="0"/>
                <xsd:element name="Def" type="xsd:double" minOccurs="0"/>
                <xsd:element name="Max" type="xsd:double" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.8 Enum

Description

Enum is a type representing an enumerable value.

Members

Value	The enumeration values.

Schema

```
<xsd:complexType name="Enum">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyA">
            <xsd:sequence>
                <xsd:element name="StateValue" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:complexContent>
                            <xsd:extension base="cn:StateValueA">
                                <xsd:sequence>
                                    <xsd:element name="Value" type="xsd:string" maxOccurs="unbounded"/>
                                </xsd:sequence>
                            </xsd:extension>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.9 EnumField

Description

Enum field is a type for enum field values.

Members

Value	Defines the field values.

Schema

```
<xsd:complexType name="EnumField">
    <xsd:complexContent>
        <xsd:extension base="cn:FieldA">
            <xsd:sequence>
                <xsd:element name="Value" type="xsd:string" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.10 EnumType

Description

Enum type is a class for enum value instances. Enum type defines common properties for all enum instances from this class. With enum type you can define enumeration types.

Members

Def	Defines the default value for the instance values. Used when the value is created without specifying initial value.
Value	Defines string, integer pair. The string value is used when user access the value, the integer value can be used by programs.

Schema

```
<xsd:complexType name="EnumType">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyTypeA">
            <xsd:sequence>
                <xsd:element name="Def" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Value" type="cn:EnumValue" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.11 EnumValue

Description

Enum value is a type for defining a pair of string and integer values. Used when defining enumeration values.

Members

String	Defines the string value for the enumeration.
Int	Defines the integer value for the enumeration.

Schema

```
<xsd:complexType name="EnumValue">
    <xsd:sequence>
        <xsd:element name="String" type="xsd:string"/>
        <xsd:element name="Int" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>
```

6.6.2.12 FieldA

Description

FieldA is a base type for field values.

Members

Schema

```
<xsd:complexType name="FieldA" abstract="true">
    <xsd:sequence/>
</xsd:complexType>
```

6.6.2.13 FieldType

Description

Field type defines one field of a struct type. Used when defining struct type.

Members

SimpleTypeId	Defines the type of the field. Simple type is any property type except struct type.
Name	Friendly name of the field.

Schema

```
    <xsd:complexType name="FieldType">
        <xsd:sequence>
            <xsd:element name="SimpleTypeId" type="fr:Uuid"/>
            <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```

6.6.2.14 Integer

Description

Integer is a representation of integer numers as 32 bit two's complement format.

Members

Value	The integer value(s).

Schema

```
<xsd:complexType name="Integer">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyA">
            <xsd:sequence>
                <xsd:element name="StateValue" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:complexContent>
                            <xsd:extension base="cn:StateValueA">
                                <xsd:sequence>
                                    <xsd:element name="Value" type="xsd:integer" maxOccurs="unbounded"/>
                                </xsd:sequence>
                            </xsd:extension>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.15 IntegerField

Description

Integer field is a type for integer field values.

Members

Value	Defines the field values.

Schema

```
<xsd:complexType name="IntegerField">
    <xsd:complexContent>
        <xsd:extension base="cn:FieldA">
            <xsd:sequence>
                <xsd:element name="Value" type="xsd:integer" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.16 IntegerType

Description

Integer type is a class for integer value instances. Integer type defines common properties for all integer instances from this class.

Members

Min	Defines the allowed minimum value for the instance values.
Def	Defines the default value for the instance values. Used when the value is created without specifying initial value.
Max	Defines the alloved maximum value for the instance values.

Schema

```
<xsd:complexType name="IntegerType">
    <xsd:complexContent>
        <xsd:extension base="cn:NumericTypeA">
            <xsd:sequence>
                <xsd:element name="Min" type="xsd:integer" minOccurs="0"/>
                <xsd:element name="Def" type="xsd:integer" minOccurs="0"/>
                <xsd:element name="Max" type="xsd:integer" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.17 ModelLibrary

Description

Model library is a grouping element. It is used to divide Models into groups.

Members

Name	Friendly name of the library.

Schema

```
<xsd:complexType name="ModelLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="ModelLibrary" type="cn:ModelLibrary" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="PlantModel" type="cn:PlantModel" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.18 NumericTypeA

Description

Numeric type is a base type for all numerical types.

Members

Unit	Unit of the type. This should be modelled better; there has been an idea that this could be modelled by Qualifier concept.

Schema

```
<xsd:complexType name="NumericTypeA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyTypeA">
            <xsd:sequence>
                <xsd:element name="Unit" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.19 ObjectA

Description

ObjectA is a base type for plant models and objects.

Members

Schema

```
<xsd:complexType name="ObjectA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="cn:SubjectA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.20 ObjectTypeA

Description

ObjectTypeA is a base type for plant model type and plant object type.

Members

Kind	Groups the objects by functionality.
BaseId	The uuid of the base class. A derived class has a union of the rules of the base and derived class. This is true even if a same base class if given twice by different inheritance hierarchies. Cyclic dependencies are not allowed.

ObjectTypeAKind

Conceptual	The object represents a conceptual view of the plant.
Functional	The object represents a functional view of the plant.
Product	The object represents a generic view of the plant.
Individiual	The object represents a physical view of the plant.

Schema

```
<xsd:complexType name="ObjectTypeA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="cn:TypeA">
            <xsd:sequence>
                <xsd:element name="Kind" type="cn:ObjectTypeAKind" minOccurs="0"/>
                <xsd:element name="BaseId" type="fr:Uuid" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="ObjectTypeAKind">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Conceptual"/>
        <xsd:enumeration value="Functional"/>
        <xsd:enumeration value="Product"/>
        <xsd:enumeration value="Individual"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.6.2.21 PlantModel

Description

Plant model represents the process plant.

Members

Final	If true then the plant model is final. A final plant model is so called runtime model, i.e. the structure of the model is freezed, but the model can be calculated so the state(s) of the model can change (i.e. the calculated property values can change).
LoadState	Records the state of the plant models load extension, if any.

LoadState

NotLoaded	The plant model does not have an load extension.
Loaded	The plant model has been loaded.
Changed	The plant model has been changed after last save.
Failed	There has been an error in the load mechanism.

Schema

```
<xsd:complexType name="PlantModel">
    <xsd:complexContent>
        <xsd:extension base="cn:ObjectA">
            <xsd:sequence>
                <xsd:element name="Final" type="xsd:boolean" minOccurs="0"/>
                <xsd:element name="State" type="cn:State" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="LoadState" type="cn:LoadState" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="LoadState">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="NotLoaded"/>
        <xsd:enumeration value="Loaded"/>
        <xsd:enumeration value="Changed"/>
        <xsd:enumeration value="Failed"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.6.2.22 PlantModelType

Description

PlantModelType is a class for plant models. Each plant model type instance has the same behaviour.

Members

Schema

```
<xsd:complexType name="PlantModelType">
    <xsd:complexContent>
        <xsd:extension base="cn:ObjectTypeA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.23 PlantObject

Description

Plant object represents building block of a plant model.

Members

Schema

```
<xsd:complexType name="PlantObject">
    <xsd:complexContent>
        <xsd:extension base="cn:ObjectA">
            <xsd:sequence>
                <xsd:element ref="ext:ParadigmReference" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.24 PlantObjectType

Description

PlantObjectType is a class for plant objects. Each plant object type instance has the same behaviour.

Members

Schema

```
<xsd:complexType name="PlantObjectType">
    <xsd:complexContent>
        <xsd:extension base="cn:ObjectTypeA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.25 PropertyA

Description

PropertyA is a base type for all properties.

Members

Schema

```
<xsd:complexType name="PropertyA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="cn:SubjectA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.26 PropertyTypeA

Description

PropertyTypeA is a base type for all property types.

Members

Size	Size of the property value vector. Zero means unbounded, one means the property is a scalar value.
Access	Access rights for the property when accessed through opc da.

PropertyTypeAAccess

None	No access.
Read	Read access.
Write	Write access.
ReadWrite	Both read and write access.

Schema

```
<xsd:complexType name="PropertyTypeA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="cn:TypeA">
            <xsd:sequence>
                <xsd:element name="Size" type="xsd:unsignedInt" minOccurs="0"/>
                <xsd:element name="Access" type="cn:PropertyTypeAAccess" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="PropertyTypeAAccess">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="None"/>
        <xsd:enumeration value="Read"/>
        <xsd:enumeration value="Write"/>
        <xsd:enumeration value="ReadWrite"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.6.2.27 Relation

Description

Relation defines a semantic relationship between two subjects.

Members

TypeId	Uuid of the relation type.

Schema

```
<xsd:complexType name="Relation">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="TypeId" type="fr:Uuid"/>
                <xsd:element name="SubjectA" type="cn:SubjectA"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.28 RelationConstraint

Description

RelationConstraint is used to restrict the more general RelationRule by enumerating the allowed end types of the parent RelationRule. Typically the relation rule specifies the allowed start type, although it is not necessary, it can define the allowed start class only.

Members

TypeId	Uuid of the allowed end type. Must conform to the class defined by the parent RelationRule, if any.
Name	Friendly name of the constraint.
Necessity	Identifies the behaviour of the corresponding subject in instantiation (when creating a start subject).
Multiplicity	Defines how many relations can be created for one start subject by this constraint. Zero means unbounded. Can only further restrict the constraint set by relation rule.

Necessity

Required	Required means that the client must give a value for corresponding relation and end subject at instantiation.
Implied	Implied means that the server creates a default instances (generating ids for the instances) for the relation and end subject if not given by the client.
Optional	Optional means that the client does not have to give give a value for corresponding relation and end subject at instantiation, and that the corresponding relation and end subject is not created.

Schema

```
<xsd:complexType name="RelationConstraint">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="TypeId" type="fr:Uuid"/>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Necessity" type="cn:Necessity" minOccurs="0"/>
                <xsd:element name="Multiplicity" type="xsd:unsignedInt" minOccurs="0"/>
                <xsd:element ref="ext:ParadigmVariable" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="Necessity">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Required"/>
        <xsd:enumeration value="Implied"/>
        <xsd:enumeration value="Optional"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.6.2.29 RelationRule

Description

RelationRules define the allowed relations between subjects. Each relation type is made of an ordered set of rules. A relation is allowed between subjects if there is a one ore more rules allowing it.

Members

SubjectType1	Defines the class of objects allowed for the start subject.
Multiplicity1	Defines how many start objects are allowed for one end object.
TypeId1	Defines the type id of the start object. If defined than SubjectType1 is deduced from TypeId1.
SubjectType2	Defines the class of objects allowed for the end subject.
Multiplicity2	Defines how many end objects are allowed for one start object.

RelationRuleType

Subject	The relation start or end can be any type derived from SubjectA
Object	The relation start or end can be any type derived from ObjectA
Property	The relation start or end can be any type derived from PropertyA

Schema

```
<xsd:complexType name="RelationRule">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="SubjectType1" type="cn:RelationRuleType" minOccurs="0"
default="Subject"/>
                <xsd:element name="Multiplicity1" type="xsd:unsignedInt" minOccurs="0" default="0"/>
                <xsd:element name="TypeId1" type="fr:Uuid" minOccurs="0"/>
                <xsd:element name="SubjectType2" type="cn:RelationRuleType" minOccurs="0"
default="Subject"/>
                <xsd:element name="Multiplicity2" type="xsd:unsignedInt" minOccurs="0" default="0"/>
                <xsd:element name="RelationConstraint" type="cn:RelationConstraint" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="RelationRuleType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Subject"/>
        <xsd:enumeration value="Object"/>
        <xsd:enumeration value="Property"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.6.2.30 RelationType

Description

Relation type defines a semantic relationship between subjects.

Members

Cyclic	It true then the graph formed from all the rules of a relation type can be from cycles.
Kind	Defines semantic class for the type.
ViewId	Uuid of a view this relation type belongs to.
RuleId	Uuid of a relation rule this relation conforms to.

RelationTypeKind

Unknonw	Nothing is known about the type.
ConsistOf	The relation represents a part whole relationship and indicates containment.
Connection	The relationship represents a connection and indicates physical connection of some kind.

```
<xsd:complexType name="RelationType">
    <xsd:complexContent>
        <xsd:extension base="cn:TypeA">
            <xsd:sequence>
                <xsd:element name="Cyclic" type="xsd:boolean" minOccurs="0" default="false"/>
                <xsd:element name="Kind" type="cn:RelationTypeKind" minOccurs="0" default="Unknown"/>
                <xsd:element name="ViewId" type="fr:Uuid" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="RuleId" type="fr:Uuid" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="RelationTypeKind">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Unknown"/>
        <xsd:enumeration value="ConsistOf"/>
        <xsd:enumeration value="Connection"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.6.2.31 State

Description

State represents a state of the model. Each state has the same structure, but the property values of each state can be different.

Members

Name	Friendly name of the state.
ParadigmReference	Used to record the non default paradigm and extension configurations.

Schema

```
<xsd:complexType name="State">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="ParadigmReference" type="ext:ParadigmReference" minOccurs="0"
    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.32 StateValueA

Description

StateValueA is a base class for all StateValues of the concrete property classess.

Members

StateId	Uuid of the state the contained values belong to.

Schema

```
<xsd:complexType name="StateValueA" abstract="true">
    <xsd:sequence>
        <xsd:element name="StateId" type="fr:Uuid"/>
    </xsd:sequence>
</xsd:complexType>
```

6.6.2.33 String

Description

String is a class representing string values.

Members

Value	The string values.

Schema

```
<xsd:complexType name="String">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyA">
            <xsd:sequence>
                <xsd:element name="StateValue" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:complexContent>
                            <xsd:extension base="cn:StateValueA">
                                <xsd:sequence>
                                    <xsd:element name="Value" type="xsd:string" maxOccurs="unbounded"/>
                                </xsd:sequence>
                            </xsd:extension>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.34 StringField

String field is a type for string field values.

Members

Value	Defines the field values.

Schema

```
<xsd:complexType name="StringField">
    <xsd:complexContent>
        <xsd:extension base="cn:FieldA">
            <xsd:sequence>
                <xsd:element name="Value" type="xsd:string" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.35 StringType

Description

String type is a class for string value instances. String type defines common properties for all instances from this class.

Members

Def	Defines the default value for the instance values. Used when the value is created without specifying initial value.

Schema

```
<xsd:complexType name="StringType">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyTypeA">
            <xsd:sequence>
                <xsd:element name="Def" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.36 Struct

Description

Struct is a user defined type, made of collection of fields that can be of any other property type except struct.

Members

Value	The struct values.

Schema

```
<xsd:complexType name="Struct">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyA">
            <xsd:sequence>
                <xsd:element name="StateValue" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:complexContent>
                            <xsd:extension base="cn:StateValueA">
```

```
                                <xsd:sequence>
                                    <xsd:element name="Value" type="cn:StructValue"
maxOccurs="unbounded"/>
                                </xsd:sequence>
                            </xsd:extension>
                        </xsd:complexContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.37 StructType

Description

Struct type is a class for struct instances.

Members

Schema

```
<xsd:complexType name="StructType">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyTypeA">
            <xsd:sequence>
                <xsd:element name="FieldType" type="cn:FieldType" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.38 StuctValue

Description

Struct value is a type for struct fields.

Members

Schema

```
<xsd:complexType name="StructValue">
    <xsd:sequence>
        <xsd:element name="FieldA" type="cn:FieldA" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

6.6.2.39 SubjectA

Description

SubjectA is a base type for all subjects. Subjects are concepts that can have relations.

Members

TypeId	Uuid of the type of the subject. Each subject has a type, representing the common features of all the subjects of same type.
Name	Friendly name for the subject.

Reference	Readonly. When given and true, the subject is already given somewere else in the response. Used to differienciate the original and the references.

Schema

```
<xsd:complexType name="SubjectA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="TypeId" type="fr:Uuid"/>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Reference" type="xsd:boolean" default="false" minOccurs="0"/>
                <xsd:element name="SubjectA" type="cn:SubjectA" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="Relation" type="cn:Relation" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.40 TypeA

Description

TypeA is a base type for all types. Types represent the class part in a class/instance kind of relationship between the types and the instances of those types.

Members

Name	Friendly name of the type.
Final	If not true the type can be modified but there can not be any instances of it. When true the type can not be modified, but can be instantiated.

Schema

```
<xsd:complexType name="TypeA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Final" type="xsd:boolean" minOccurs="0" default="false"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.41 TypeLibrary

Description

Type library is a grouping element. It is used to divide types into logical groups.

Members

Name	Friendly name of the library.

Schema

```
<xsd:complexType name="TypeLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="View" type="cn:View" minOccurs="0" maxOccurs="unbounded"/>
```

```
                <xsd:element name="TypeA" type="cn:TypeA" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="TypeLibrary" type="cn:TypeLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.42 View

Description

View is a mechanism to define different logical views of the whole process plant model. Each view is made of relation types, by following those relations client can be presented a different logial view of the whole process plant model.

Members

Name	Friendly name of the view.

Schema

```
<xsd:complexType name="View">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element ref="g2d:Svg"  minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="g2d:Symbol"  minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="g3d:Model3D" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.43 ModelLibrary

Description

Model library is a grouping element. It is used to divide Models into groups.

Members

Name	Friendly name of the library.

Schema

```
<xsd:complexType name="ModelLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="ModelLibrary" type="cn:ModelLibrary" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="PlantModel" type="cn:PlantModel" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.44 Void

Description

Void is a class representing a valueless property.

Members

Schema

```
<xsd:complexType name="Void">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.45 VoidField

Void field is a type for valueless fields.

Members

Schema

```
<xsd:complexType name="VoidField">
    <xsd:complexContent>
        <xsd:extension base="cn:FieldA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.2.46 VoidType

Description

Void type is a class for valueless instances. Void type defines common properties for all instances from this class.

Members

Schema

```
<xsd:complexType name="VoidType">
    <xsd:complexContent>
        <xsd:extension base="cn:PropertyTypeA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.6.3 Operations

6.6.3.1 GetRelationType

Description

Get a relation type by view and subjects.

6.6.3.1.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
ViewId	Identifies the view to use.
FromSubjectId	Identifies the start subject.
ToSubjectId	Identifies the end subject.

Schema

```
<xsd:element name="GetRelationTypeRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="ViewId" type="fr:Uuid"/>
            <xsd:element name="FromSubjectId" type="fr:Uuid" minOccurs="0"/>
            <xsd:element name="ToSubjectId" type="fr:Uuid" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.6.3.1.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.
RelationType	The matching relation types.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="GetRelationTypeResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
            <xsd:element name="RelationType" type="cn:RelationType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7 Extension

6.7.1 Introduction

This chapter explains the extension types. The normative description of the types is in extension.xsd.

6.7.2 Types

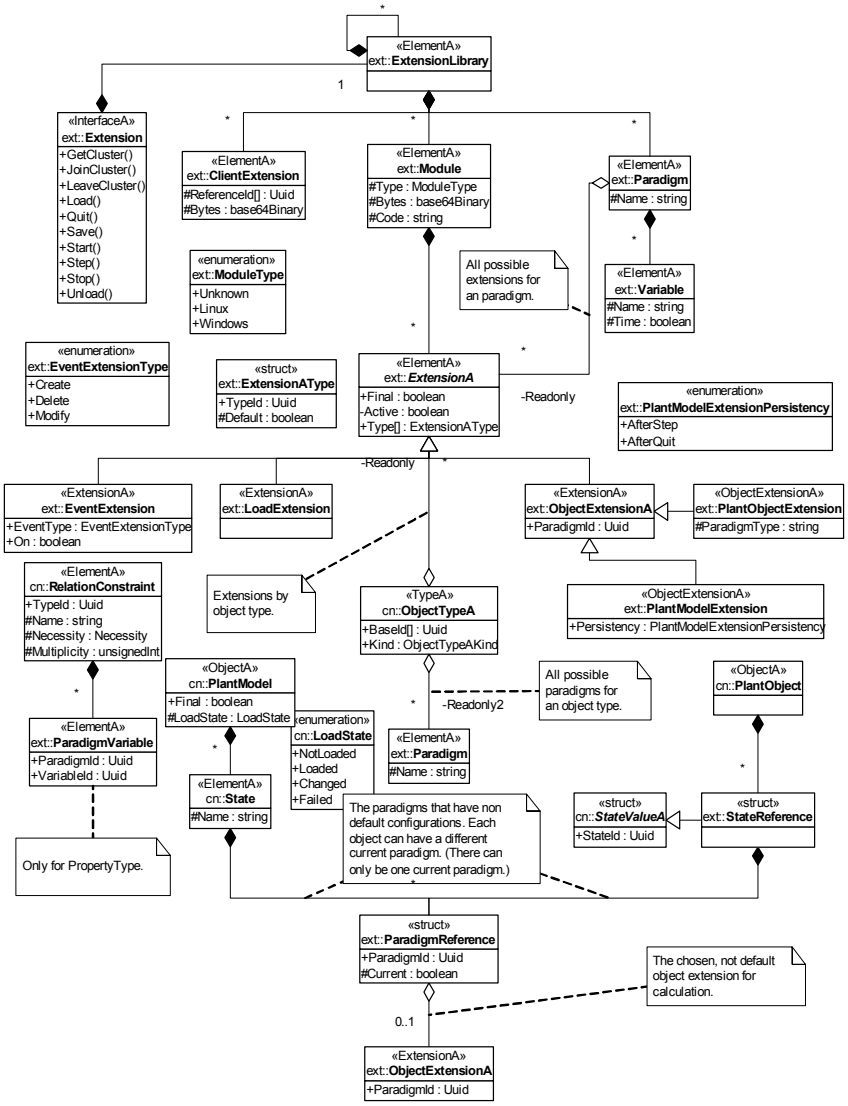


Figure 6-7. Types of extension interface.

6.7.2.1 ClientExtension

Description

Clients can extend the features that are available for users with a client extension. Client extension is a component that is run on the client machine, so it's not really an extension at all; it's here only because of the similarity of the terms. Its invocation mechanism is client specific, and client applications must know which client extensions they support and what they are for.

Members

ReferenceId	List of elements this client extension is associated with.
Bytes	The bytes of the client extension.

Schema

```
<xsd:complexType name="ClientExtension">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="ReferenceId" type="fr:Uuid" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="Bytes" type="xsd:base64Binary" minOccurs="0"/>
            </xsd:sequence>
    </xsd:extension>
</xsd:complexContent>
```

6.7.2.2 EventExtension

Description

Event extension is an extension that is called when a plant object is cretated, deleted, or modified.

Members

Type	Type of event.
On	True if extension is activated, i.e. is called when event happens.

EventExtensionType

Create	Object created.
Modify	Object modified.
Delete	Object deleted.

Schema

```
<xsd:complexType name="EventExtension">
    <xsd:complexContent>
        <xsd:extension base="ext:ExtensionA">
            <xsd:sequence>
                <xsd:element name="Type" type="ext:EventExtensionType"/>
                <xsd:element name="On" type="xsd:boolean" default="false"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="EventExtensionType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Create"/>
        <xsd:enumeration value="Modify"/>
        <xsd:enumeration value="Delete"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.7.2.3 Extension

Description

Extension is the type representing the corresponding interface.

Members

Schema

```
<xsd:complexType name="Extension">
    <xsd:complexContent>
        <xsd:extension base="fr:InterfaceA">
            <xsd:sequence>
                <xsd:element name="ClientExtension" type="ext:ClientExtension" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="Module" type="ext:Module" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="Paradigm" type="ext:Paradigm" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="ExtensionLibrary" type="ext:ExtensionLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.4 ExtensionA

Description

ExtensionA is a base class for extensions. Extension is a component that extends the functionality of the service. When used the module containing the extension is loaded. If the extension instance is not yet created, it is created before the extension instance object is called. Each extension type has a different interface. Extension can not be deleted when in use. Extension can be deleted when event is done (EventExtension), when unload is done (LoadExtension), or when quit is done (PlantModel and object Extensions).

Members

Final	If true the extension can not be modified. If false the extension can not be used in calculation of property values.
Active	If true the extension is in use (the module loaded and the extension implementation created).

Schema

```
<xsd:complexType name="ExtensionA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Final" type="xsd:boolean" default="false"/>
                <xsd:element name="Active" type="xsd:boolean" default="true" minOccurs="0"/>
                <xsd:element name="Type" type="ext:ExtensionAType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.5 ExtensionAType

Description

ExtensionAType is a structure for recording the supported object types of an extension. Only on extension can be a default extension for an object type.

Members

TypeId	Defines the object type the parent Extension supports.
Default	If true than this is the default extension for the given object type.

Schema

```
<xsd:complexType name="ExtensionAType">
    <xsd:sequence>
        <xsd:element name="TypeId" type="fr:Uuid"/>
        <xsd:element name="Default" type="xsd:boolean" default="false"/>
    </xsd:sequence>
</xsd:complexType>
```

6.7.2.6 ExtensionLibrary

Description

Grouping element, used to divide the extensions into logical groups.

Members

Schema

```
<xsd:complexType name="ExtensionLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Paradigm" type="ext:Paradigm" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="ExtensionA" type="ext:ExtensionA" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="ClientExtension" type="ext:ClientExtension" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="ExtensionLibrary" type="ext:ExtensionLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.7 LoadExtension

Description

Load extension provides the load functionality for models. Load extension is a component that is run on the server machine. The load mechanism is for implementations that keep the structure of the model (i.e. root component) or parts of it in internal representation (legacy systems and speed optimization).

The load implementation must dynamically build or update so much of the plant hierarchy to the server than it wants to open to the client when the model is loaded. The

extension gets the plant model object and is offered about the same services for modifying the model as the web clients, i.e. configuration and extension interfaces. If the structure or property values of the loadable model or its child objects are changed, a notification function is called and the implementation may modify its internal structures accordingly and/or return error code/texts for the operation that caused the change notification.

Note that the load implementation is responsible of the state the model is left. If the client has done illegal modifications it is up to the load implementation to correct these if necessary. When the model is unloaded its load implementation can clean up as it sees fit. It can leave the stuff under the model because when a model is loadable, its internal structure is not shown to clients when the model's load extensioin is not loaded, so the structure and property values do not have to be up to date when the implementation is not loaded. However, the implementation must bring them up to date when the model is loaded.

Implementation Module(s) must fulfill the normal Module requirements (initialize, finalize). In addition, load extension must have the following logical structure:

- LoadExtension (singleton): create, delete, make_component.
- LoadComponent: delete, load, save, unload, changed.
- Load: Called when the model is loaded via the load operation.
- Save: Called when the component was saved via the save operatioin.
- Unload: Called when the component was unloaded via the unload operation.
- Changed: Called when the model has changed.

Members

Schema

```
<xsd:complexType name="LoadExtension">
    <xsd:complexContent>
        <xsd:extension base="ext:ExtensionA">
            <xsd:sequence/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.8 Module

Description

Module represents a dynamically loadable shared library that can provide implementations of extensions. Each module can have the following bookkeeping routines:

- Initialize: Called after the module is loaded and before first extension is called.
- Finalize: Called after the last extension has finished and before the module is unloaded. Must release all resources so that the module can be unloaded.

Comments

The problem with the Extension definition is that some malicious user can make an Extension that will hog all the resources and make the service unusable.

What is the working diretory of an extension? Legasy systems may store countless of files relative to working directory. Should this be definable? And more importantly should the disk space usage be controlled?

What is the extension security model? The whole idea of allowing binary/script components to be set and run is inherently unsecure, unwise and should not be allowed under any excuse! At least there should be some mechanism to verify that Extensions are written by a responsible party before running them should be even considered.

Members

Type	The requirements for the usage of the module.
Bytes	Implementation bytes, mutually exclusive with code.
Code	Implementation code, mutually exclusive with bytes.

ExtensionType

Unknown	The module is binary independent.
Linux	Linux shared library, kernel version 2.4, libgcc 3.2.
Windows	Windows XP dynamically loadable library, .Net 2003 C++.

Schema

```
<xsd:complexType name="Module">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Type" type="ext:ModuleType" minOccurs="0" default="Unknown"/>
                <xsd:element name="Bytes" type="xsd:base64Binary" minOccurs="0"/>
                <xsd:element name="Code" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="ModuleType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Unknown"/>
        <xsd:enumeration value="Linux"/>
        <xsd:enumeration value="Windows"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.7.2.9 ObjectExtensionA

Description

ObjectExtensionA is a base type for all extensions that need a paradigm.

Members

ParadigmId	Uuid of the paradigm this extension supports.

Schema

```
<xsd:complexType name="ObjectExtensionA">
    <xsd:complexContent>
        <xsd:extension base="ext:ExtensionA">
            <xsd:sequence>
                <xsd:element name="ParadigmId" type="fr:Uuid"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.10 Paradigm

Description

Paradigm defines common rules for building plant model and object extensions to form a coherent calculation model of the process plant model.

Members

Name	Friendly name for the paradigm.
ExtensionA	This element is used to return all possible extensions for an Paradigm.

Schema

```
<xsd:complexType name="Paradigm">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Variable" type="ext:Variable" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="ExtensionA" type="ext:ExtensionA" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.11 ParadigmReference

Description

Paradigm reference records the non default configuration changes the user has made to the current plant model and object extensions.

Members

ParadigmId	Uuid of the paradigm
Current	If true than this is the current paradigm
ObjectExtensionA	The chosen, not default object extension for calculation.

Schema

```
<xsd:complexType name="ParadigmReference">
    <xsd:sequence>
        <xsd:element name="ParadigmId" type="fr:Uuid"/>
        <xsd:element name="Current" type="xsd:boolean" minOccurs="0" default="false"/>
        <xsd:element name="ObjectExtensionA" type="ext:ObjectExtensionA" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

6.7.2.12 ParadigmVariable

Description

Paradigm variable represents a mapping of property type and/or property to paradigm concept. Paradigm concept is a set of rules covering how the process plant model is modelled and calculated under the given paradigm.

Members

ParadigmId	Uuid of the paradigm
VariableId	Uuid of the paradigm variable.

Schema

```
<xsd:complexType name="ParadigmVariable">
    <xsd:sequence>
        <xsd:element name="ParadigmId" type="fr:Uuid"/>
        <xsd:element name="VariableId" type="fr:Uuid"/>
    </xsd:sequence>
</xsd:complexType>
```

6.7.2.13 PlantModelExtension

Description

PlantModelExtension provides the simulation functionality for models. PlantModelExtension is a component that is run on the server machine. The run mechanism is for implementing the quit, start, step, and stop operations. The service will call the implemention before first step, for each step, and after quit. If client uses start operation the service will step continuously until stop operation is given or the step returns error or time does not advance.

The implementation may register addresses for properties it might modify, or whose modifications (by some other client through CN or DA interfaces) it wants to follow. During the step the implementation can then change the values at these addresses. Between steps the implementation must not read or write to these values because the service might be using them. At the next step the implementation may check if the service has changed these values. At quit the implementation must free all the resources it has acquired.

The proposed implementation strategy of plant model extensions is such that the server calls the plant model extension implementation as simple function calls, and vise verse, but for the plant model extension implementator a wrapping from function model to an

object model is done. The implementator must implement and function to create plant model extension object (a singleton). The plant model extension object must be able to create instances (i.e. component objects) and to be able to delete itself. The plant model component (i.e. the instance) must be able to respond to prepare, step, and delete operations. Prepare is called before the step is called first time, and every time a value not registred changes, or a structural changes is made. Step is called when the simulation time should advance one tick. Quit is called when the service has no need for the component. (Changes destroying registered values are not allowed.)

Members

Persistency	The persistency model, see below. Relevant only if the interface data is persistent (versus save/load model).

PlantModelExtensionPersistency

AfterStep	The changed values are updated to the persistent store after every step.
AfterQuit	The changed values are updated to the persistent store only after quit.

Schema

```
<xsd:complexType name="PlantModelExtension">
    <xsd:complexContent>
        <xsd:extension base="ext:ObjectExtensionA">
            <xsd:sequence>
                <xsd:element name="Persistency" type="ext:PlantModelExtensionPersistency"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="PlantModelExtensionPersistency">
<xsd:simpleType name="PlantModelExtensionPersistency">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="AfterStep"/>
        <xsd:enumeration value="AfterQuit"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.7.2.14 PlantObjectExtension

Description

PlantObjectExtension provides the unit operation for model parts. PlantObjectExtension is a component that is run on the server machine. The mechanism is for implementing binary independent calculation operation. The service will call the implemention's calculation operation under the direction of the plant model simulation engine (plant model extension).

PlantObjectExtension implementation must have the following logical routines:

- PlantObjectExtension (singleton): create, make_component, delete
- PlantObjectComponent: calculate, delete.

Members

ParadigmType	Type of object extension. Can be used by the paradigm to type its objects extensions, e.g. unit operation, solver, property databank. The meaning of this is paradigm specific, and can be found from the paradigm documentation.

Schema

```
<xsd:complexType name="PlantObjectExtension">
    <xsd:complexContent>
        <xsd:extension base="ext:ObjectExtensionA">
            <xsd:sequence>
                <xsd:element name="ParadigmType" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.15 StateReference

Description

State reference represents the non default paradigm and extension configurations.

Members

Schema

```
<xsd:complexType name="StateReference">
    <xsd:complexContent>
        <xsd:extension base="cn:StateValueA">
            <xsd:sequence>
                <xsd:element name="ParadigmReference" type="ext:ParadigmReference" minOccurs="0"
    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.2.16 Variable

Description

Variable defines the necessary properties for a paradigm.

Members

Name	Friendly name for the variable.
Time	If true than the variable is a time variable. A paradigm can only have a one time variable. Time variable provides a calculated time concept for the process plant model. When a paradigm has a time variable it must be mapped to a property by a relation rule (see paradigm variable), must be instantiated and referred by the plant model, and the property's type must be double. If the time does not advance when the plant model is stepped, then the calculation is considered to fail. If a paradigm does not have a time variable, the service will provide a real time time stamps for the property values when needed.

Schema

```
<xsd:complexType name="Variable">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Time" type="xsd:boolean" minOccurs="0" default="false"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.7.3 Operations

6.7.3.1 GetCluster

Description

The command gets the id and name of the cluster the model state is part of, if any.

6.7.3.1.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
StateId	The uuid of the state. Each state can be part of only one cluster.

Schema

```
<xsd:element name="GetClusterRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="StateId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.1.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.
ClusterId	Identifies the cluster the state is part of.
Name	Name of the cluster.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="GetClusterResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
```

```
            <xsd:element name="ClusterId" type="fr:Uuid"  minOccurs="0"/>
            <xsd:element name="Name" type="xsd:string"  minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.2 JoinCluster

Description

Join the given state to the given cluster. When state is part of a cluster, a calculation advances one tick and then stops. It then sends responses to all other members of the cluster which have pending polledRefresh requests (or have subscribed FDA callbacks) and waits until it has received responses to its own polledRefresh requests to the other cluster members (or FDA callbacks). Then it advances another tick.

When cluster member subscribes data, it tells the server its cluster (in the OPC XML DA Subscribe request and/or FDA Subscribe request). This way each cluster member can identify which subscriptions are part of the cluster and thus will be part of the synchronization mechanism. The behavior of DA polledRefresh is changed in a cluster so that wait and hold times do not have any meaning and the server will always wait one tick and use zero hold time. The behavior of FDA callback is changed in a cluster so that tick has no meaning and the server will always use the cluster tick and not the FDA subscription request tick.

The join cluster request means that the OPC server must re-subscribe all its data (the OPC Server does not know which of the other servers are part of the cluster). For the mechanism to work, the client must first stop all cluster servers, then do join for each of them, and then start each of them. The initial data is changed after the start, so the first polledRefesh calls must return before the first tick so that the initial state is well defined.

Comments

Jyrki thought better that the synchronization mechansim would be separate, should it could be used in as many cases as possible.

6.7.3.2.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
StateId	Identifies the state.
ClusterId	Identifies the cluster.
Tick	How much to advance before synchronization.
Name	Friendly name for the cluster.

Schema

```
<xsd:element name="JoinClusterRequest">
    <xsd:complexType>
```

```
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="StateId" type="fr:Uuid"/>
            <xsd:element name="ClusterId" type="fr:Uuid"/>
            <xsd:element name="Tick" type="xsd:double"/>
            <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.2.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_EXIST	Element or some other entity does not exist.

Schema

```
    <xsd:element name="JoinClusterResponse">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="fr:Result"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
```

6.7.3.3 LeaveCluster

Description

Leave a cluster the state has joined previously.

6.7.3.3.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
StateId	Identifies the state.
ClusterId	Identifies the cluster.

Schema

```
<xsd:element name="LeaveClusterRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="StateId" type="fr:Uuid"/>
            <xsd:element name="ClusterId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.3.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="LeaveClusterResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.4 LoadModel

Description

Load the model. If a model is loadable, it must be loaded before it can be started.

6.7.3.4.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
ModelId	Identifies the model to load.

Schema

```
<xsd:element name="LoadModelRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="ModelId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.4.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.
E_NOT_LOADABLE	The element is not loadable.

Schema

```
<xsd:element name="LoadModelResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.5 Quit

Description

Quit calculating a property values for a state. The instance of the plant model extension is deleted, and the plant model object extension instances are also deleted. If the extension instance and extension (class) objects are the last in the module, it is also unloaded.

6.7.3.5.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
StateId	Identifiest the state to quit.

Schema

```
<xsd:element name="QuitRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="StateId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.5.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="QuitResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.6 SaveModel

Description

Save the model. If model is loadable it must be saved or the changes are not stored.

6.7.3.6.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
ModelId	Identifies the model to save.

Schema

```
<xsd:element name="SaveModelRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="ModelId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.6.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="SaveModelResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.7 Start

Description

Start calculating the property values of the model for the given state.

6.7.3.7.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
StateId	Identifies the state to calculate.

Schema

```
<xsd:element name="StartRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="StateId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.7.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="StartResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.8 Step

Description

Advance the calculation one step.

6.7.3.8.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
StateId	Identifies the state to step.

Schema

```
<xsd:element name="StepRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="StateId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.8.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="StepResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.9 Stop

Description

Stop calculating the property values started with start operation. Not same as quit, which destroyes any run time data structures.

6.7.3.9.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
StateId	Identifies the state to stop.

Schema

```
<xsd:element name="StopRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="StateId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.9.2 Response

Arguments

Result	Context is the common first argument for most of the framework requests.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="StopResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.10 UnloadModel

Description

Unload the model. If component is loadable it must be unload. Unload does not save changes made since last save. Unload can free a considerable amount of system resources.

6.7.3.10.1 Request

Arguments

Context	Context is the common first argument for most of the framework requests.
ModelId	Identifies the model to unload.

Schema

```
<xsd:element name="UnloadModelRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Context" type="fr:Context" minOccurs="0"/>
            <xsd:element name="ModelId" type="fr:Uuid"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.7.3.10.2 Response

Arguments

Result	Result is the common return type for most framework responses. Note that errors are given as SOAP exceptions, not as result codes.

Errors

E_FAIL	Operation failed partly or completely.
E_REQUEST	Request did not conform to the wsdl.
E_ILLEGAL_ARGUMENT	Although confirming request, one or more argument values were illegal for the operation.

Schema

```
<xsd:element name="UnloadModelResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="fr:Result"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.8 Graphic 2D

6.8.1 Introduction

This chapter explains the two dimensional graphics interface. The normative description of the types is in g2d.xsd. The interface is based to Scalable Vector Graphics XML specifications of W3C. The standard elements are extended by framework, and graphics 2d namespace elements, and attributes. Here are documented only the extended attributes. The extended svg elements are svg, symbol, g, defs, and use.

6.8.2 Types

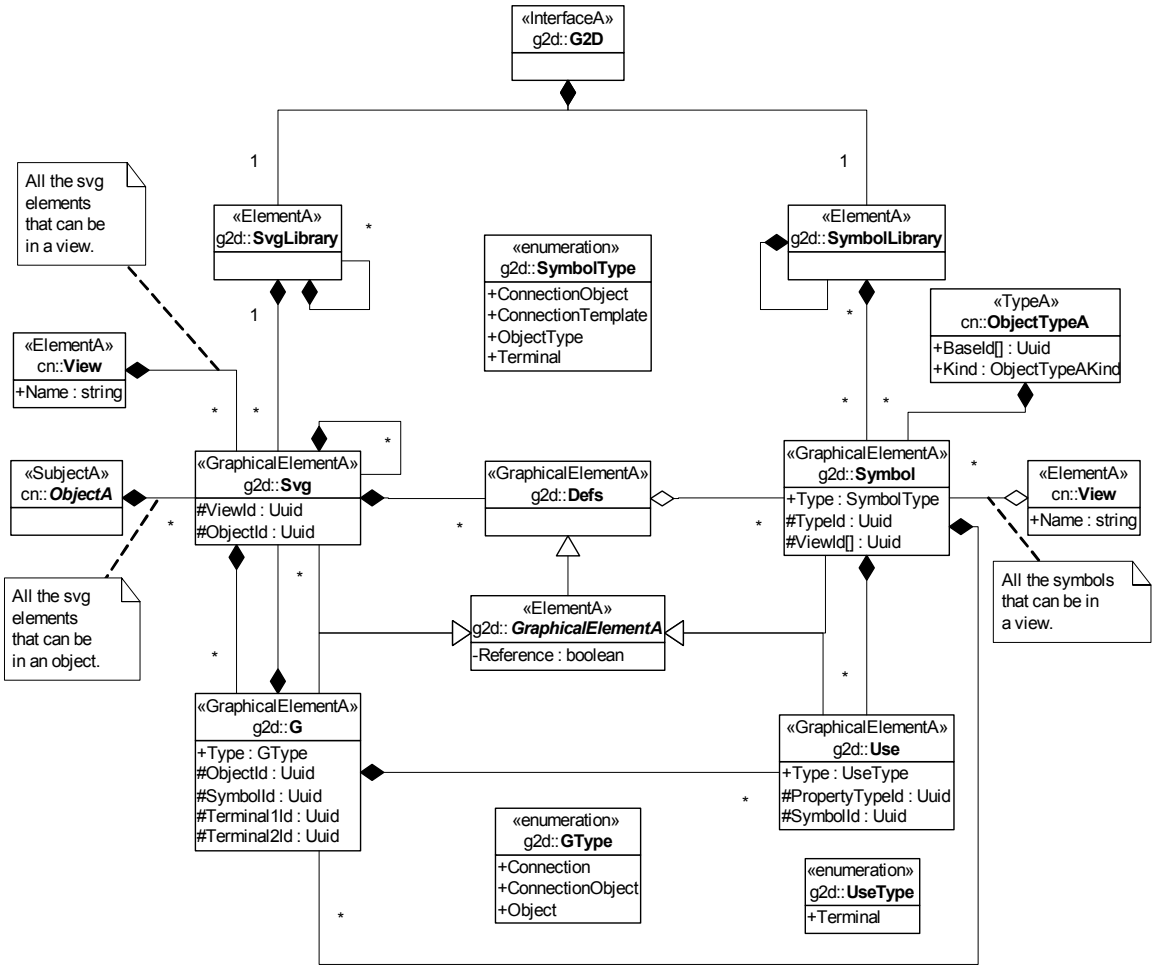


Figure 6-8. Types of the graphic 2d interface.

6.8.2.1 Defs

Description

Defs element represents the graphical definitions that can be used in the containing svg element.

Members

Type	Type of the element.

Schema

```
<xsd:complexType name="Defs">
    <xsd:complexContent>
        <xsd:extension base="g2d:GraphicalElementA">
            <xsd:sequence>
                <xsd:element name="Symbol" type="g2d:Symbol" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.8.2.2 G

Description

G element represents a one semantic graphical aggregate of the ancestor svg or symbol element. The svg elements under the g element are used to shift the coordinate system of the containing graphical representation. In addition the svg elements are needed if defs elements are used.

Members

Type	Type of the element.
ObjectId	Uuid of the object the element represents. Valid if type is Object or ConnectionObject.
SymbolId	Uuid of the symbol the element represents. Valid if type is ConnectionObject, and in this case the symbol's type is ConnectionObject.
Terminal1Id	Uuid of the object or property that represents the first terminal in a connection. Valid if type is Connection or ConnectionObject.
Terminal2Id	Uuid of the object or property that represents the second terminal in a connection. Valid if type Connection or ConnectionObject.

GType

Connection	Represents a connection from first terminal to second terminal. Connection in the configuration interface corresponds to a relation between two subjects.
ConnectionObject	Represents a connection from two terminals to two named terminals of a given object. In the configuration interface corresponds to two relations and an object.
Object	Represents an object.

Schema

```
<xsd:complexType name="G">
    <xsd:complexContent>
        <xsd:extension base="g2d:GraphicalElementA">
            <xsd:sequence>
                <xsd:element name="Type" type="g2d:GType"/>
                <xsd:element name="ObjectId" type="fr:Uuid" minOccurs="0"/>
                <xsd:element name="SymbolId" type="fr:Uuid" minOccurs="0"/>
                <xsd:element name="Terminal1Id" type="fr:Uuid" minOccurs="0"/>
                <xsd:element name="Terminal2Id" type="fr:Uuid" minOccurs="0"/>
                <xsd:element name="Use" type="g2d:Symbol" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="Svg" type="g2d:Svg" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="GType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Connection"/>
        <xsd:enumeration value="ConnectionObject"/>
        <xsd:enumeration value="Object"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.8.2.3 G2D

Description

G2D is the type representing the two dimensional graphical interface.

Members

Schema

```
<xsd:complexType name="G2D">
    <xsd:complexContent>
        <xsd:extension base="fr:InterfaceA">
            <xsd:sequence>
                <xsd:element name="SymbolLibrary" type="g2d:SymbolLibrary" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="SvgLibrary" type="g2d:SymbolLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.8.2.4 GraphicalElementA

Description

GraphicalElementA is a base class for all graphical elements.

Members

Reference	Readonly. When given and true, the subject is already given somewere else in the response. Used to differienciate the original and the references.

Schema

```
<xsd:complexType name="GraphicalElementA" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
```

```
            <xsd:sequence>
                <xsd:element name="Reference" type="xsd:boolean" minOccurs="0"/>
                <xsd:any namespace="##any" processContents="lax" minOccurs="0"
    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:anyAttribute namespace="##any" processContents="lax"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.8.2.5 Svg

Description

Svg represents a view of an object's internals. It presents a two dimensional view of the objects in relation to the given object.

Members

ViewId	Identifies to which view this svg element belongs.
ObjectId	Identifies the object this view represents.

Schema

```
<xsd:complexType name="Svg">
    <xsd:complexContent>
        <xsd:extension base="g2d:GraphicalElementA">
            <xsd:sequence>
                <xsd:element name="ViewId" type="fr:Uuid"/>
                <xsd:element name="ObjectId" type="fr:Uuid"/>
                <xsd:element name="Defs" type="g2d:Defs" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="G" type="g2d:G" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SvgLibrary">
```

6.8.2.6 SvgLibrary

Description

Grouping element, used to divide the svg elements into logical groups.

Members

Schema

```
<xsd:complexType name="SvgLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Svg" type="g2d:Svg" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="SvgLibrary" type="g2d:SvgLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.8.2.7 Symbol

Description

Symbol is a description of graphical symbols of connections, objects, and terminals. The same symbol can be referred from many defs elements. If get must return the same Symbol more than once in any given response, it returns the original only once, and for the rest only references (see GraphicalElementA).

Members

Type	Type of the symbol.
TypeId	Uuid of the object type if the type is ObjectType, else uuid of the property type. Valid if type is ObjectType, Terminal, or Terminal.
ViewId	List of views this symbol can be used in.

SymbolType

ConnectionObject	The symbol is used as a template when generating graphical representation of and object representating a connection between two terminals. This is a combined construct of an object and two connections between the object's two named terminals and two other terminals.
ConnectionTemplate	The symbol is used as a template when generating graphical representation of reference from from first terminal to second terminal.
ObjectType	The symbol is used as a graphical representation of an Object.
Terminal	The symbol is a graphical representation of Terminal.

Schema

```
<xsd:complexType name="Symbol">
    <xsd:complexContent>
        <xsd:extension base="g2d:GraphicalElementA">
            <xsd:sequence>
                <xsd:element name="Type" type="g2d:SymbolType"/>
                <xsd:element name="TypeId" type="fr:Uuid" minOccurs="0"/>
                <xsd:element name="ViewId" type="fr:Uuid" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="Use" type="g2d:Use" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="G" type="g2d:G" minOccurs="0"   maxOccurs="unbounded"/>
            </xsd:sequence>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="SymbolType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ConnectionObject"/>
        <xsd:enumeration value="ConnectionTerminal"/>
        <xsd:enumeration value="ObjectType"/>
        <xsd:enumeration value="Terminal"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.8.2.8 SymbolLibrary

Description

Grouping element, used to divide the symbols into logical groups.

Members

Schema

```
<xsd:complexType name="SymbolLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Symbol" type="g2d:Symbol" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="SymbolLibrary" type="g2d:SymbolLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.8.2.9 Use

Description

Use element is a graphical representation of a terminal or object.

Members

Type	Type of the element.
TypeId	Identifies the type of the Terminal. Can be property type id or object type id. Valid if type is Terminal.
SymbolId	Identifies the symbol. If use element is under symbol, then the Symbol's type must be Terminal. If the use element is under g element, then the Symbol's type must be ObjectType.

UseType

Object	Use of object symbol.
Terminal	Use of terminal symbol.

Schema

```
<xsd:complexType name="Use">
    <xsd:complexContent>
        <xsd:extension base="g2d:GraphicalElementA">
            <xsd:sequence>
                <xsd:element name="Type" type="g2d:UseType"/>
                <xsd:element name="TypeId" type="fr:Uuid" minOccurs="0"/>
                <xsd:element name="SymbolId" type="fr:Uuid" minOccurs="0"/>
                <xsd:any namespace="##any" processContents="lax" minOccurs="0"
    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:anyAttribute namespace="##any" processContents="lax"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
 <xsd:simpleType name="UseType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Object"/>
        <xsd:enumeration value="Terminal"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.9 Graphic 3D

6.9.1 Introduction

This chapter explains the three dimensional graphics interface. The normative description of the interface types is in g3d.xsd.

6.9.2 Types

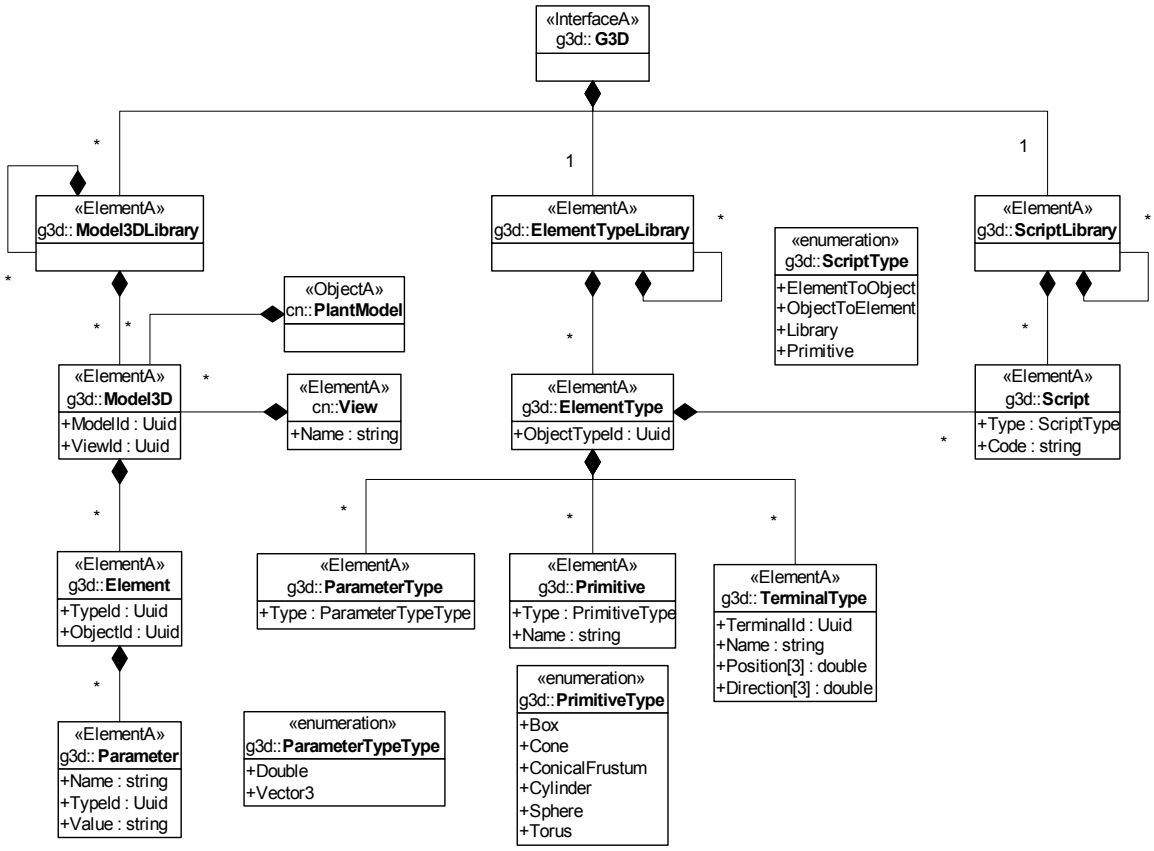


Figure 6-9. Types of graphic 3d interface.

6.9.2.1 Element

Description

Element represents a 3d element in one or more 3d views.

Members

TypeId	Uuid of the type of the element.
ObjectId	Uuid of the object the element represents.

Schema

```
<xsd:complexType name="Element">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="TypeId" type="fr:Uuid"/>
                <xsd:element name="ObjectId" type="fr:Uuid"/>
                <xsd:element name="Parameter" type="g3d:Parameter" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.9.2.2 ElementType

Description

ElementType represents the class/instance relationship between element's type and its instances.

Members

ObjectTypeId	Identifies the object type the element type corresponds to.

Schema

```
<xsd:complexType name="ElementType">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="ObjectTypeId" type="fr:Uuid"/>
                <xsd:element name="ParameterType" type="g3d:ParameterType" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="Primitive" type="g3d:Primitive" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="TerminalType" type="g3d:TerminalType" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.9.2.3 ElementTypeLibrary

Description

Grouping element, used to divide the ElementTypes into logical groups.

Members

ComponentTypeId	Identifies the ComponentType.

Schema

```
<xsd:complexType name="ElementTypeLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="ElementType" type="g3d:ElementType" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="ElementTypeLibrary" type="g3d:ElementTypeLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.9.2.4 G3D

Description

G3D is the type representing the three dimensional graphical interface.

Members

Schema

```
<xsd:complexType name="G3D">
    <xsd:complexContent>
        <xsd:extension base="fr:InterfaceA">
            <xsd:sequence>
                <xsd:element name="ElementTypeLibrary" type="g3d:ElementTypeLibrary" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="ScriptLibrary" type="g3d:ScriptLibrary" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="Model3DLibrary" type="g3d:Model3DLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Model3D">
```

6.9.2.5 Model3D

Description

Model3D represents a 3d view of the process plant model. Each view can have a corresponding 3d model.

Members

ModelId	Uuid of the corresponding process plant model.
ViewId	Uuid of the view this 3d view represents.

Schema

```
<xsd:complexType name="Model3D">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="ModelId" type="fr:Uuid"/>
                <xsd:element name="ViewId" type="fr:Uuid"/>
                <xsd:element name="Element" type="g3d:Element" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.9.2.6 Model3DLibrary

Description

Grouping element, used to divide the 3d models into logical groups.

Members

Schema

```
<xsd:complexType name="Model3DLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Model3D" type="g3d:Model3D" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="Model3DLibrary" type="g3d:Model3DLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.9.2.7 Parameter

Description

Parameter represents a parameter used to describe some feature of the parent element. The parameter can be used in the scripts doing the mapping of elements into 3d view and back.

Members

Name	Name of the paramter.
TypeId	The uuid of the parameter type.
Value	The parameter value.

Schema

```
<xsd:complexType name="Parameter">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Name" type="xsd:string"/>
                <xsd:element name="TypeId" type="fr:Uuid"/>
                <xsd:element name="Value" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.9.2.8 ParameterType

Description

ParameterType represents the class/instance relationship of parameters.

Members

Type	Type of the parameter.

ParameterTypeType

Double	The parameter is a double value.
Vector3	The parameter is a a vector of three double values (x, y, z).

Schema

```
<xsd:complexType name="ParameterType">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Type" type="g3d:ParameterTypeType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="ParameterTypeType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Double"/>
        <xsd:enumeration value="Vector3"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.9.2.9 Primitive

Description

Primitive is an instance of a 3d shape class. ElementType consists of named primitives, which can be referred from the scripts.

Members

Type	Type of the primitive.
Name	Name of the primitive.

PrimitiveType

Box	See English English dictionary.
Cone	See English English dictionary.
ConicalFrustum	Also called a truncated cone, basically a cylinder with different radiuses at each end.
Cylinder	See English English dictionary.
Sphere	See English English dictionary.
Torus	The torus is parametrized so that you can create trucated tori that can represent angular pipes.

Schema

```
<xsd:complexType name="Primitive">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Type" type="g3d:PrimitiveType"/>
                <xsd:element name="Name" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="PrimitiveType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Box"/>
        <xsd:enumeration value="Cone"/>
        <xsd:enumeration value="ConicalFrustum"/>
        <xsd:enumeration value="Cylinder"/>
        <xsd:enumeration value="Sphere"/>
        <xsd:enumeration value="Torus"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.9.2.10 Script

Description

The script is a Jython script used to map the 3d elements to and back from the form used by the 3d viewer.

Members

Type	The type of the script
Code	The code of the script.

ScriptType

ElementToObject	The script maps element modifications back to the object.
ObjectToElement	The script maps object modifications back to the element.
Library	The script is used by other scripts.
Primitive	

Schema

```
<xsd:complexType name="Script">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Type" type="g3d:ScriptType"/>
                <xsd:element name="Code" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="ScriptType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ElementToObject"/>
        <xsd:enumeration value="ObjectToElement"/>
        <xsd:enumeration value="Library"/>
        <xsd:enumeration value="Primitive"/>
    </xsd:restriction>
</xsd:simpleType>
```

6.9.2.11 ScriptLibrary

Description

Grouping element, used to divide the scripts into logical groups.

Members

Schema

```
<xsd:complexType name="ScriptLibrary">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="Script" type="g3d:Script" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="ScriptLibrary" type="g3d:ScriptLibrary" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.9.2.12 TerminalType

Description

TerminalType describes the properties of the 3d elements used to bind different 3d objects together.

Members

TerminalId	Identifies the corresponding terminal. Can be a property or an object.
Name	Name for the 3d editing system.
Position	Position of the 3d terminal object.
Direction	Direction of the 3d terminal object.

Schema

```
<xsd:complexType name="TerminalType">
    <xsd:complexContent>
        <xsd:extension base="fr:ElementA">
            <xsd:sequence>
                <xsd:element name="TerminalId" type="fr:Uuid"/>
                <xsd:element name="Name" type="xsd:string"/>
                <xsd:element name="Position" type="xsd:double" minOccurs="3" maxOccurs="3"/>
                <xsd:element name="Direction" type="xsd:double" minOccurs="3" maxOccurs="3"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

6.10 FastDataAccess

6.10.1 Introduction

Fast data access is an extension to OPC XML data access specification. The extension is done due to the special performance requirements of the dynamic process simulation. The extension is described for research and testing purposes only. If the ideas are utilized in practise the developer has to comply with copyright statement of the OPC Foundation.

6.10.2 Types

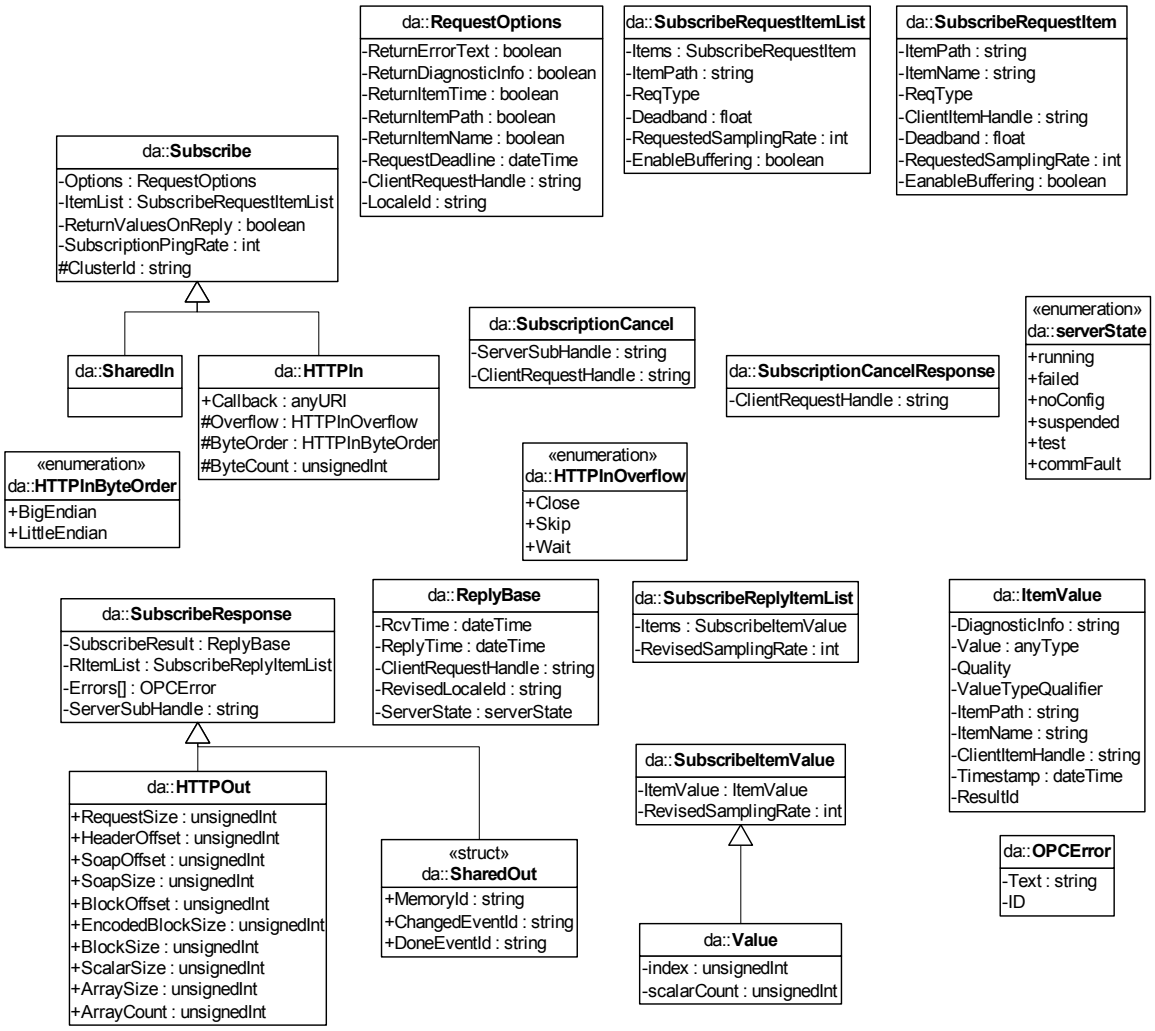


Figure 6-10. Types of fast data access interface.

Fast data access extends subscribe and cancel operations of the OPC XML DA interface. With the extended subscribe you can order transfer callback and with the extended cancel you can revoke it. If the subscribed item is buffered the transfer callback will be used to transfer the buffered values before current values so that the client does not have to do anything special to handle the buffered values. The OPC XML data access specification is modified so that the GetStatus operation returns information if the DA server is FDA compliant, and the VendorData element contains the FDA URL of the server. If the server is FDA compliant, instead of OPC XML DA, the fastest available FDA transfer method is used.

6.10.2.1 HTTPIn

Description

HttpIn is an extension to DA subscription mechanism. All subscribed and found items form a fixed sized array of data. Changed data in this array is transferred in one or more HTTP messages as block of bytes to the client using a callback mechanism. All items must be of same type and have a fixed size. The transferred block of bytes is base64Binary encoded (see CN.Callback). The block is transferred in fixed size HTTP messages as fixed size SOAP messages. The client doesn't have to parse anything, because the HTTP message is always fixed size, the SOAP message is always fixed size, and the data block is always fixed size, but the client must still decode the base64Binary encoded block and copy the transferred bytes to the subscription array. The size of the subscription array is fixed and contains only elements of same type.

The elements don't have a state because only valid items are added to the subscription array, and as long as the subscription (callback) is open all the items are valid. The server can divide the transferred bytes in continuous blocks as it whiches. The client must implement the transfer callback operation. This will be called one or more times each time the subscribed data changes. If the client is too slow the server will either break the connection, wait until the client catches up, or skip changes. In other kind of error coditions the server will break the transfer callback connection. If the transfer connection breaks for any reason the server will cancel the subscription.

Members

Callback	URI for the transfer SOAP callback. The callback is used to send the subscribed data.
Overflow	How an overflow situation is handled. Default Close.
ByteOrder	Which way the bytes are delivered. Default big endian.
ByteCount	Maximum number of data bytes per one transferred block. If zero server can decide. Default is 0.

HTTPInOverflow

Close	An overflow closes the connection.

Skip	An overflow causes the server to skip data transfer cycles until the client catches up.
Wait	An overflow causes the server to wait until the client catches up.

HTTPInByteOrder

BigEndian	Bytes ordered from left to right, i.e. 0123.
LittleEndian	Bytes ordered from right to left, i.e. 3210.

Block

Name	C++Type	Description
BaseTime	DateTime	The base time of the model.
ElapsedTime	Double	Elapsed time in seconds. Relative to the base time.
Status	Unsigned int	0 bit = last packet of this sample 1 bit = you lost samples (overflow, skip mode) 2 bit = you are too slow (overflow, wait mode)
Offset	Unsigned int	Zero based offset to the overall data array for the bytes.
Count	Unsigned int	Number of transferred bytes.
Bytes	Char	The transferred bytes.

Schema

```
<s:complexType name="HTTPIn">
    <s:complexContent>
        <s:extension base="s0:Subscribe">
            <s:attribute name="Callback" type="s:anyURI" use="required"/>
            <s:attribute name="Overflow" type="s0:HTTPInOverflow"/>
            <s:attribute name="ByteOrder" type="s0:HTTPInByteOrder"/>
            <s:attribute name="ByteCount" type="s:unsignedInt"/>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:simpleType name="HTTPInByteOrder">
    <s:restriction base="s:string">
        <s:enumeration value="BigEndian"/>
        <s:enumeration value="LittleEndian"/>
    </s:restriction>
</s:simpleType>
<s:simpleType name="HTTPInOverflow">
    <s:restriction base="s:string">
        <s:enumeration value="Close"/>
        <s:enumeration value="Skip"/>
        <s:enumeration value="Wait"/>
    </s:restriction>
</s:simpleType>
```

6.10.2.2 HTTPOut

Description

HTTPOut is an extension to OPC XML DA SubscribeResponse.

Members

RequestSize	Size of one HTTP packet. Must be big enough for HTTPIn.byteCount bytes. See FDA.HTTPIn.
HeaderOffset	Offset of the HTTP headers from start of the request.
SoapOffset	Offset of the soap block from start of the request.
SoapSize	Size of the soap block, i.e. the size of the HTTP content block.

BlockOffset	The offset for the data block from start of the request.
EncodedBlockSize	Size of the encoded data block (which does not contain white spaces).
BlockSize	Size of the decoded data block.
ScalarSize	Size of one scalar item. (Items can be scalars or vectors of same type.)
ArrayCount	Number of valid/transferred items in the subscription data array.
ArraySize	Overall size of the subscription data array. (If on or more items are vectors then this is bigger then arrayCount * ScalarSize.)

Schema

```
<s:complexType name="HTTPOut">
    <s:complexContent>
        <s:extension base="s0:SubscribeResponse">
            <s:attribute name="RequestSize" type="s:unsignedInt" use="required"/>
            <s:attribute name="HeaderOffset" type="s:unsignedInt" use="required"/>
            <s:attribute name="SoapOffset" type="s:unsignedInt" use="required"/>
            <s:attribute name="SoapSize" type="s:unsignedInt" use="required"/>
            <s:attribute name="BlockOffset" type="s:unsignedInt" use="required"/>
            <s:attribute name="EncodedBlockSize" type="s:unsignedInt" use="required"/>
            <s:attribute name="BlockSize" type="s:unsignedInt" use="required"/>
            <s:attribute name="ScalarSize" type="s:unsignedInt" use="required"/>
            <s:attribute name="ArraySize" type="s:unsignedInt" use="required"/>
            <s:attribute name="ArrayCount" type="s:unsignedInt" use="required"/>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

6.10.2.3 SharedIn

Description

SharedIn in is an extension to OPC XML DA Subscribe. The data of the subscription is updated to a shared memory. The client and server lock the memory, do their stuff and release the memory. There are no members; the only data is the type of the element (SharedIn).

Members

Schema

```
<s:complexType name="SharedIn">
    <s:complexContent>
        <s:extension base="s0:Subscribe"/>
    </s:complexContent>
</s:complexType>
```

6.10.2.4 SharedOut

Description

SharaedOut is an extension to OPC XML DA SubscribeResponse.

Members

MemoryId	Identifies the shared memory.
ChangedEventId	Identifies the event sent to the client after modifying the memory.
DoneEventId	Identifies the event sent to the server after handling the changes.

Schema

```
<s:complexType name="SharedOut">
    <s:complexContent>
        <s:extension base="s0:SubscribeResponse">
            <s:attribute name="MemoryId" type="s:string" use="required"/>
            <s:attribute name="ChangedEventId" type="s:string" use="required"/>
            <s:attribute name="DoneEventId" type="s:string" use="required"/>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

6.10.2.5 Subscribe

Description

Subscribe is an extension to OPC XML DA Subscribe element.

Members

ClusterId	Identifies the cluster the client belongs to.

Schema

```
<s:complexType name="Subscribe">
    <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Options" type="s0:RequestOptions"/>
        <s:element minOccurs="0" maxOccurs="1" name="ItemList" type="s0:SubscribeRequestItemList"/>
    </s:sequence>
    <s:attribute name="ReturnValuesOnReply" type="s:boolean" use="required"/>
    <s:attribute default="0" name="SubscriptionPingRate" type="s:int"/>
    <s:attribute name="ClusterId" type="s:string"/>
</s:complexType>
```

6.10.2.6 Value

Description

Value is an extension to OPC XML DA SubscribeItemValue.

Members

Index	Zero based index to the overall subscription data array or shared memory for the item in question.
ScalarCount	Number of scalars for this item.

Schema

```
<s:complexType name="Value">
    <s:complexContent>
        <s:extension base="s0:SubscribeItemValue">
            <s:attribute name="Index" type="s:unsignedInt" use="required"/>
            <s:attribute name="ScalarCount" type="s:unsignedInt" use="required"/>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

6.10.3 Operations

6.10.3.1 Transfer

Description

A callback the server will use to transfer changes to subscription array. See FDA.HTTPIn. No response expected!

6.10.3.1.1 Request

Arguments

Bytes	The transferred block, see HTTPIn and HTTPOut.

Schema

```
<xsd:element name="TransferRequest">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Bytes" type="xsd:base64Binary" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

6.11 DataExchange

The behaviour of a FDA compliant core service implementing OPC data exchange specification is modified so that if the configured SourceServer is also FDA compliant, instead of OPC XML DA, the fastest FDA transfer method is used.

Acknowledgements

The Proserv project was funded by the National Technology Agency (Tekes), ÅF-CTS, Fortum Nuclear Services, Metso Automation, Alma Software and VTT Industrial Systems. During two years 10 research scientists and trainees from VTT Industrial Systems, Helsinki University of Technology and Tampere University of Technology have participated to the project.

References

ACE. 2004. The Adaptive Communication Environment by Douglas C. Schmidt and his research group at Washington, http://www.cs.wustl.edu/~schmidt/ACE.html, [referenced 2004-12-20].

Apache. 2004. Axis 1.1, http://ws.apache.org/axis/index.html, [referenced 2004-12-27].

Castor. 2004. Castor 0.9.5.3, http://castor.exolab.org/, [referenced 2004-12-27].

DCE 1998. Distributed Computing Environment. UUID draft specification. http://www.opengroup.org/dce/info/draft-leach-uuids-guids-01.txt

Expat. 1999. The Expat XML Parser by James Clark, http://www.libexpat.org/, [referenced 2004-12-20].

IEEE. 2000. IEEE Std 1471-2000.

Oasis. 2002. Web Services Security (WS-Security), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, [referenced 2004-12-20].

Salkari, I., Seuranen, T., Viinikkala, M., Pötry, J., Karhela, T. 2004. Information Management in Process Plant Life Cycle: Case Fibre Refining. VTT Report.

W3C. 1997. Hypertext Transfer Protocol – HTTP/1.1", R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, T. Berners-Lee, January 1997, http://www.w3.org/Protocols/rfc2616/rfc2616.html , [referenced 2004-12-20].

W3C. 2000. Extensible Markup Language, http://www.w3.org/XML/, [referenced 2004-12-20].

W3C. 2001. Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl, [referenced 2004-12-20].

W3C. 2002a. XML Encryption Syntax and Processing, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/, [referenced 2004-12-20].

W3C. 2002b. XML Signature Syntax and Processing, http://www.w3.org/TR/xmldsig-core/, [referenced 2004-12-20].

W3C. 2003. SOAP Simple Object Access Protocol (SOAP) 1.2, http://www.w3.org/TR/soap/, [referenced 2004-12-20].

Published by

Series title, number and
report code of publication

VTT Research Notes 2277
VTT–TIED–2277

Author(s)
Kondelin, Kalle, Karhela, Tommi & Laakso, Pasi

Title

Service Framework Specification for Process Plant Lifecycle

Abstract

This specification descibes a software infrastructure for storing, deploying and using process plant information through the lifecycle of the process plant from design to construction and usage. Document describes the architecture and interfaces of the infrastucture. The specification was written in a research project (2003-2004) funded by National Technology Agency (Tekes), Finnish industry and VTT. This document is one of the end products of the ProServ project.

Keywords
industrial plants, industrial processes, plant models, value-added services, web services, service networks, life cycle evaluation, maintenance, simulation, information management

Activity unit
VTT Industrial Systems, Tekniikantie 12, P.O.Box 1301, FIN-02044 VTT, Finland

ISBN	Project number
951–38–6521-5 (soft back ed.) 951–38–6522-3 (URL: http://www.vtt.fi/inf/pdf/)	

Date	Language	Pages	Price
January 2005	English	123 p.	C

Name of project	Commissioned by
Proserv project	National Technology Agency (Tekes), ÅF-CTS, Fortum Nuclear Services, Metso Automation, Alma Software, VTT Industrial Systems

Series title and ISSN	Sold by
VTT Tiedotteita – Research Notes 1235–0605 (soft back edition) 1455–0865 (URL: http://www.vtt.fi/inf/pdf/)	VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 20 722 4404 Fax +358 20 722 4374

VTT TIEDOTTEITA – RESEARCH NOTES

VTT TUOTTEET JA TUOTANTO – VTT INDUSTRIELLA SYSTEM –
VTT INDUSTRIAL SYSTEMS

2208 Räikkönen, Timo & Rouhiainen, Veikko. Riskienhallinnan muutosvoimat. Kirjallisuuskatsaus. 2003. 77 p.

2216 Savioja, Paula. Käyttäjäkeskeiset menetelmät monimutkaisten järjestelmien vaatimusten kuvaamisessa. 2003. 132 s. + liitt. 10 s.

2225 Tuotannonohjaus pk-konepajateollisuuden alihankintaprosessissa. Käytäntöjä suomalaisessa pk-konepajateollisuudessa vuonna 2003. 2003. 82 s.

2228 Kettunen, Jari & Reiman, Teemu. Ulkoistaminen ja alihankkijoiden käyttö ydinvoimateollisuudessa. 2004. 66 s. + liitt. 2 s.

2231 Häkkinen, Tarja, Vares, Sirje & Siltanen, Pekka. Tuotteiden käyttöikäinformaatio ja sen käyttö rakennushankkeessa. 2004. 54 s. + liitt. 32 s.

2232 Pötry, Jyri, Kettunen, Outi & Kilponen, August. Varaston ulkoistaminen alihankinnassa. Kustannusmallitarkastelu. 2004. 57 s. + liitt. 22 s.

2233 Hyötyläinen, Raimo, Ryynänen, Tapani & Mikkola, Markku. Ympäristöalan miniklustereiden rakentaminen ja kehittäminen. InnoEnvi-hanke. 2004. 111 s.

2235 Lehto, Taru & Murtonen, Mervi. Toiminnan kehittämisen vaikutukset ja päätöksenteko. PRIMA-työkalupakki kehittämistoimenpiteiden valintaan ja suunnitteluun. 2004. 52 s. + liitt. 35 s.

2240 Jarimo, Toni. Innovation Incentives in Enterprise Networks. A Game Theoretic Approach. 2004. 63 p. + app. 3 p.

2243 Ventä, Olli. Älykkäät palvelut -teknologiatiekartta. 2004. 71 s. + liitt. 11 s.

2250 Sippola, Merja, Brander, Timo, Calonius, Kim, Kantola, Lauri, Karjalainen, Jukka-Pekka, Kortelainen, Juha, Lehtonen, Mikko, Söderström, Patrik, Timperi, Antti & Vessonen, Ismo. Funktionaalisten materiaalien mahdollisuudet lujitemuovisessa toimirakenteessa. 2004. 216 s.

2251 Riikonen, Heli, Valkokari, Katri & Kulmala, Harri I. Palkitseminen kilpailukyvyn parantajana. Tuotantopalkkauksen kehittämismenetelmät vaatetusalalla. 2004. 67 s.

2254 Nuutinen, Maaria. Etäasiantuntijapalvelun haasteet. Työn toiminta- ja osaamisvaatimusten mallintaminen. 2004. 31 s.

2257 Koivisto, Tapio, Lehto, Taru, Poikkimäki, Jyrki, Valkokari, Katri & Hyötyläinen, Raimo. Metallin ja koneenrakennuksen liiketoimintayhteisöt Pirkanmaalla. 2004. 33 s.

2263 Pöyhönen, Ilkka & Hukki, Kristiina. Riskitietoisen ohjelmiston vaatimusmäärittelyprosessin kehittäminen. 2004. 36 s. + liitt. 9 s.

2264 Malm, Timo & Kivipuro, Maarit. Turvallisuuteen liittyvät ohjausjärjestelmät konesovelluksissa. Esimerkkejä. 2004. 90 s. + liitt. 4 s.

2265 Alanen, Jarmo, Hietikko, Marita & Malm, Timo. Safety of Digital Communications in Machines. 2004. 93 p. + app. 1 p.

2269 Mikkola, Markku, Ilomäki, Sanna-Kaisa & Salkari, Iiro. Uutta liiketoimintaa osaamista yhdistämällä. 2004. 65 s.

2277 Kondelin, Kalle, Karhela, Tommi & Laakso, Pasi. Service Framework Specification for Process Plant Lifecycle. 2004. 123 p.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU PL 2000 02044 VTT Puh. 020 722 4404 Faksi 020 722 4374	VTT INFORMATIONSTJÄNST PB 2000 02044 VTT Tel. 020 722 4404 Fax 020 722 4374	VTT INFORMATION SERVICE P.O.Box 2000 FIN–02044 VTT, Finland Phone internat. + 358 20 722 4404 Fax + 358 20 722 4374

ISBN 951–38–6521–5 (soft back ed.) ISBN 951–38–6522–3 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1235–0605 (soft back ed.) ISSN 1455–0865 (URL: http://www.vtt.fi/inf/pdf/)