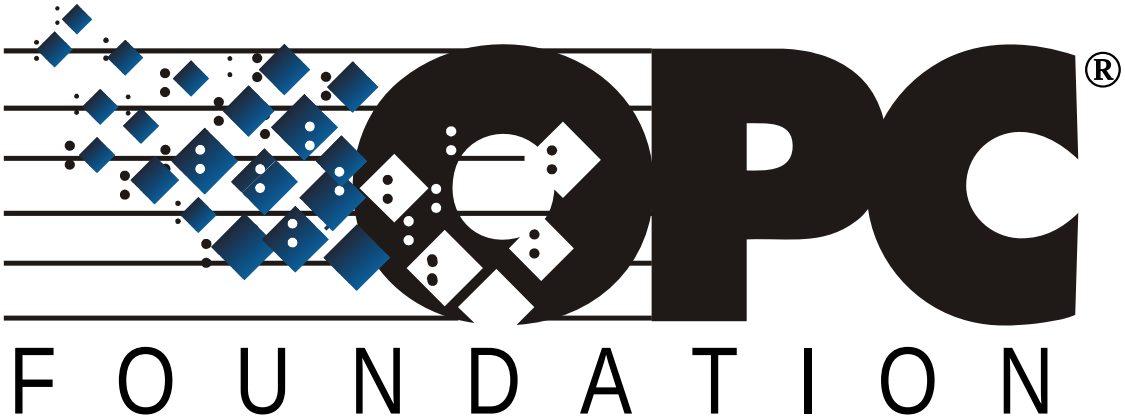

OPC Unified Architecture

Specification

Part 1: Concepts

Version 1.00

July 28, 2006

CONTENTS

1 Scope

This specification presents an overview on the OPC Unified Architecture concepts.

2 Reference documents

The OPC Unified Architecture Specification is organized as a multi-part document. While describing the concepts, this part will refer to these parts of the specification:

[UA Part 2] OPC UA Specification: Part 2 – Security Model, Version 1.0 or later
http://www.opcfoundation.org/UA/Part2/

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model, Version 1.0 or later
http://www.opcfoundation.org/UA/Part3/

[UA Part 4] OPC UA Specification: Part 4 – Services, Version 1.0 or later
http://www.opcfoundation.org/UA/Part4/

[UA Part 5] OPC UA Specification: Part 5 – Information Model, Version 1.0 or later
http://www.opcfoundation.org/UA/Part5/

[UA Part 6] OPC UA Specification: Part 6 – Mappings, Version 1.0 or later
http://www.opcfoundation.org/UA/Part6/

[UA Part 7] OPC UA Specification: Part 7 – Profiles, Version 1.0 or later
http://www.opcfoundation.org/UA/Part7/

[UA Part 8] OPC UA Specification: Part 8 – Data Access, Version 1.0 or later
http://www.opcfoundation.org/UA/Part8/

[UA Part 9] OPC UA Specification: Part 9 – Alarms and Conditions, Version 1.0 or later
http://www.opcfoundation.org/UA/Part9/

[UA Part 10] OPC UA Specification: Part 10 – Programs, Version 1.0 or later
http://www.opcfoundation.org/UA/Part10/

[UA Part 11] OPC UA Specification: Part 11 – Historical Access, Version 1.0 or later
http://www.opcfoundation.org/UA/Part11/

3 Terms, definitions, and abbreviations

For the purposes of this specification, the following definitions apply.

3.1 OPC UA terms

3.1.1 AddressSpace

The collection of information that an OPC UA *Server* makes visible to its *Clients*. See [UA Part 3] for a description of the contents and structure of the *Server AddressSpace*.

3.1.2 Alarm

A type of *Event* associated with a state condition that typically requires acknowledgement. See [UA Part 9] for a description of *Alarms*.

3.1.3 Attribute

A primitive characteristic of a *Node*. All *Attributes* are defined by OPC UA, and may not be defined by *Clients* or *Servers*. *Attributes* are the only elements in the *AddressSpace* permitted to have data values.

3.1.4 Certificate

A digitally signed data structure that describes the capabilities of a *Client* or *Server*.

3.1.5 Client

A software application that sends *Messages* to OPC UA *Servers* conforming to the *Services* specified in this set of specifications.

3.1.6 Communication Stack

A layered set of software modules between the application and the hardware that provides various functions to encode, encrypt and format a *Message* for sending, and to decode, decrypt and unpack a *Message* that was received.

3.1.7 Complex Data

Data that is composed of elements or more than one primitive data type, such as a structure.

3.1.8 Event

A generic term used to describe an occurrence of some significance within a system or system component.

3.1.9 EventNotifier

A special *Attribute* of a *Node* that signifies that a *Client* may subscribe to that particular *Node* to receive *Notifications* of *Event* occurrences.

3.1.10 Information Model

An organizational framework that defines, characterizes and relates information resources of a given system or set of systems. The core address space model supports the representation of *Information Models* in the *AddressSpace*. See [UA Part 5] for a description of the base OPC UA *Information Model*.

3.1.11 Message

The data unit conveyed between *Client* and *Server* that represents a specific *Service* request or response.

3.1.12 Method

A callable software function.

3.1.13 MonitoredItem

A *Client*-defined entity in the *Server* used to monitor *Attributes* or *EventNotifiers* for new values or *Event* occurrences and generate *Notifications* for them.

3.1.14 Node

The fundamental component of an *AddressSpace*.

3.1.15 NodeClass

The class of a *Node* in an *AddressSpace*. *NodeClasses* define the metadata for the components of the OPC UA Object Model. They also define constructs, such as *Views,* that are used to organize the *AddressSpace*.

3.1.16 Notification

The generic term for data that announces the detection of an *Event* or of a changed *Attribute* value. *Notifications* are sent in *NotificationMessages*.

3.1.17 NotificationMessage

A *Message* published from a *Subscription* that contains one or more *Notifications*.

3.1.18 Object

A *Node* that represents a physical or abstract element of a system. *Objects* are modelled using the OPC UA Object Model. Systems, subsystems and devices are examples of *Objects*. An *Object* may be defined as an instance of an *ObjectType*.

3.1.19 Object Instance

A synonym for *Object*. Not all *Objects* are defined by *ObjectTypes*.

3.1.20 ObjectType

A *Node* that represents the type definition for an *Object*.

3.1.21 Profile

A specific set of capabilities, defined in [UA Part 7], to which a *Server* may claim conformance. Each *Server* may claim conformance to more than one *Profile*.

3.1.22 Program

An executable *Object* that, when invoked, immediately returns a response to indicate that execution has started, and then returns intermediate and final results through *Subscriptions* identified by the *Client* during invocation.

3.1.23 Reference

An explicit relationship (a named pointer) from one *Node* to another. The *Node* that contains the *Reference* is the source *Node*, and the referenced *Node* is the target *Node*. All *References* are defined by *ReferenceTypes*.

3.1.24 ReferenceType

A *Node* that represents the type definition of a *Reference*. The *ReferenceType* specifies the semantics of a *Reference*. The name of a *ReferenceType* identifies how source *Nodes* are related to target *Nodes* and generally reflects an operation between the two, such as “A *Contains* B”.

3.1.25 RootNode

The beginning or top *Node* of a hierarchy. The *RootNode* of the OPC UA *AddressSpace* is defined in [UA Part 5].

3.1.26 Server

A software application that implements and exposes the *Services* specified in this set of specifications.

3.1.27 Service

A *Client*-callable operation in an OPC UA *Server. Services* are defined in [UA Part 4]. A *Service* is similar to a method call in a programming language or an operation in a Web services WSDL contract.

3.1.28 Service Set

A group of related *Services*.

3.1.29 Session

A logical long-running connection between a *Client* and a *Server.* A *Session* maintains state information between *Service* calls from the *Client* to the *Server*.

3.1.30 Subscription

A *Client*-defined endpoint in the *Server,* used to return *Notifications* to the *Client*. Generic term that describes a set of *Nodes* selected by the *Client* (1) that the *Server* periodically monitors for the existence of some condition, and (2) for which the *Server* sends *Notifications* to the *Client* when the condition is detected.

3.1.31 Variable

A *Variable* is a *Node* that contains a value.

3.1.32 View

A specific subset of the *AddressSpace* that is of interest to the *Client*.

3.2 Abbreviations and symbols

A&E	Alarms and Events
API	Application Programming Interface
COM	Component Object Model
DA	Data Access
DX	Data Exchange
HDA	Historical Data Access
HMI	Human-Machine Interface
MES	Manufacturing Execution System
PLC	Programmable Logic Controller

SCADA	Supervisory Control And Data Acquisition
SOAP	Simple Object Access Protocol
UA	Unified Architecture
UML	Unified Modelling Language
WSDL	Web Services Definition Language
XML	Extensible Mark-up Language

4 Structure of the OPC UA series

4.1 Specification Organization

This specification is organized as a multi-part specification, as illustrated in Figure 1.

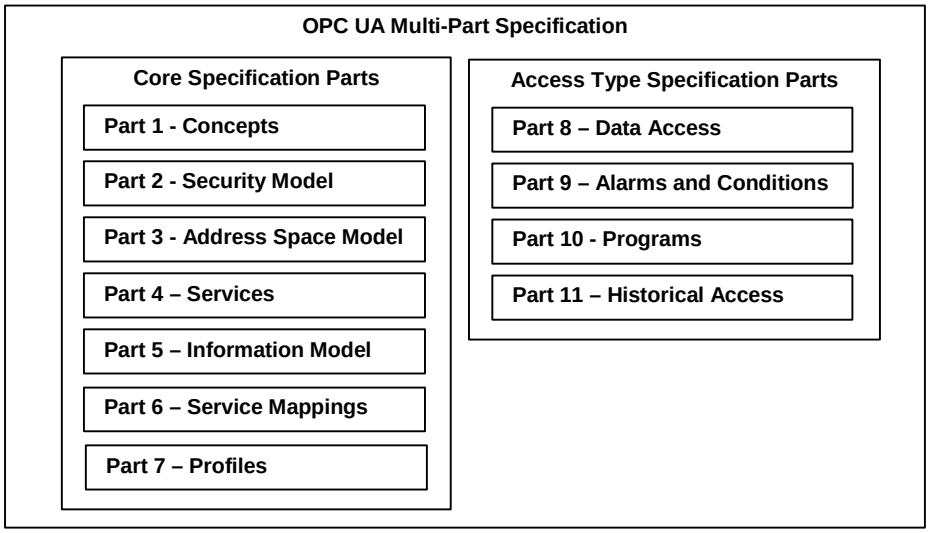

Figure 1 – OPC UA Specification Organization

The first seven parts specify the core capabilities of OPC UA. These core capabilities define the structure of the OPC *AddressSpace* and the *Services* that operate on it. Parts 8 through 11 apply these core capabilities to specific types of access previously addressed by separate OPC COM specifications, such as Data Access (DA), Alarms and Events (A&E) and Historical Data Access (HDA).

Readers are encouraged to read Parts 1 through 5 of the core specifications before reading Parts 8 through 11. For example, a reader interested in UA Data Access should read Parts 1 through 5 and 8. References in [UA Part 8] may direct the reader to other parts of this specification.

4.2 Core Specification Parts

4.2.1 Part 1 – OPC UA Concepts

This specification describes the concepts applicable to OPC UA *Servers* and *Clients*.

4.2.2 Part 2 – OPC UA Security Model

[UA Part 2] describes the model for securing interactions between OPC UA *Clients* and OPC UA *Servers*.

4.2.3 Part 3 – OPC UA Address Space Model

[UA Part 3] describes the contents and structure of the *Server's AddressSpace*.

4.2.4 Part 4 – OPC UA Services

[UA Part 4] specifies the *Services* provided by OPC UA *Servers*.

4.2.5 Part 5 – OPC UA Information Model

[UA Part 5] specifies the standard types and their relationships defined for OPC UA *Servers*.

4.2.6 Part 6 – OPC UA Service Mappings

[UA Part 6] specifies the transport mappings and data encodings supported by OPC UA.

4.2.7 Part 7 – OPC UA Profiles

[UA Part 7] specifies the *Profiles* that are available for OPC *Clients* and *Servers*. These *Profiles* provide groups of *Services* or functionality that can be used for conformance level certification. *Servers* and *Clients* will be tested against the *Profiles*.

4.3 Access Type Specification Parts

4.3.1 Part 8 – OPC UA Data Access

[UA Part 8] specifies the use of OPC UA for data access.

4.3.2 Part 9 – OPC UA Alarms and Conditions

[UA Part 9] specifies use of OPC UA support for access to *Alarms* and conditions. The base system includes support for simple *Events*; this specification extends that support to include support for *Alarms* and conditions.

4.3.3 Part 10 – OPC UA Programs

[UA Part 10] specifies OPC UA support for access to *Programs*.

4.3.4 Part 11 – OPC UA Historical Access

[UA Part 11] specifies use of OPC UA for historical access. This access includes both historical data and historical *Events*.

5 Overview

5.1 Introduction

OPC Unified Architecture (UA) is a platform-independent standard through which various kinds of systems and devices can communicate by sending *Messages* between *Clients* and *Servers* over various types of networks. It supports robust, secure communication that assures the identity of *Clients* and *Servers* and resists attacks. OPC UA defines standard sets of *Services* that *Servers* may provide, and individual *Servers* specify to *Clients* what *Service* sets they support. Information is conveyed using standard and vendor-defined data types, and *Servers* define object models that *Clients* can dynamically discover. *Servers* can provide access to both current and historical data, as well as *Alarms* and *Events* to notify *Clients* of important changes. OPC UA can be mapped onto a variety of communication protocols and data can be encoded in various ways to trade off portability and efficiency.

5.2 Design goals

OPC UA provides a consistent, integrated *AddressSpace* and service model. This allows a single OPC UA *Server* to integrate data, *Alarms* and *Events*, and history into its *AddressSpace*, and to provide access to them using an integrated set of *Services*. These *Services* also include an integrated security model.

OPC UA also allows *Servers* to provide *Clients* with type definitions for the *Objects* accessed from the *AddressSpace*. This allows standard information models to be used to describe the contents of

the *AddressSpace*. OPC UA allows data to be exposed in many different formats, including binary structures and XML documents. The format of the data may be defined by OPC, other standard organizations or vendors. Through the *AddressSpace*, *Clients* can query the *Server* for the metadata that describes the format for the data. In many cases, *Clients* with no pre-programmed knowledge of the data formats will be able to determine the formats at runtime and properly utilize the data.

OPC UA adds support for many relationships between *Nodes* instead of being limited to just a single hierarchy. In this way, an OPC UA *Server* may present data in a variety of hierarchies tailored to the way a set of *Clients* would typically like to view the data. This flexibility, combined with support for type definitions, makes OPC UA applicable to a wide array of problem domains. As illustrated below, OPC UA is not targeted at just the telemetry server interface, but also as a way to provide greater interoperability between higher level functions.

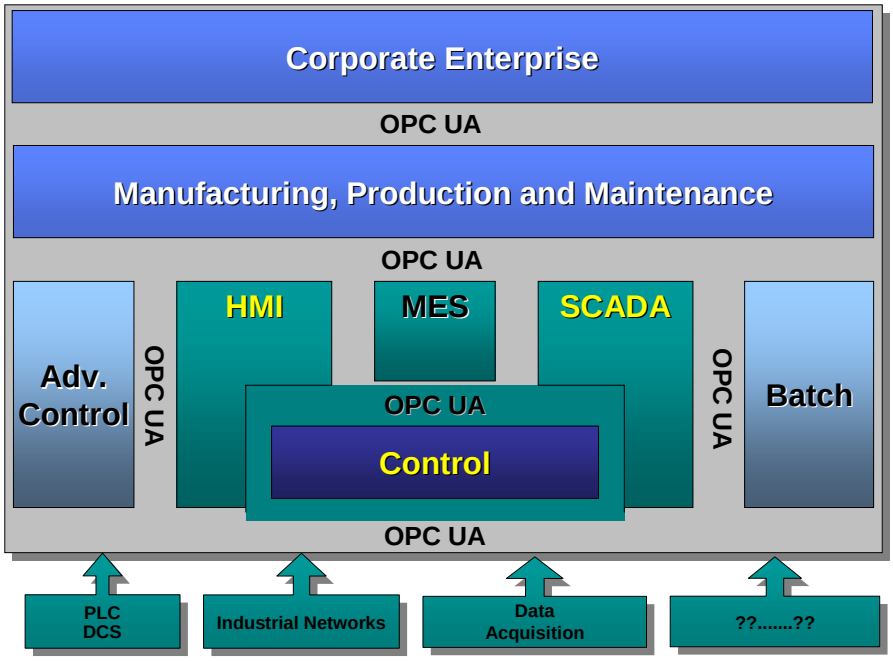

Figure 2 – OPC UA Target Applications

OPC UA is designed to provide robustness of published data. A major feature of all OPC servers is the ability to publish data and *Event Notifications*. OPC UA provides mechanisms for *Clients* to quickly detect and recover from communication failures associated with these transfers without having to wait for long timeouts provided by the underlying protocols.

OPC UA is designed to support a wide range of *Servers*, from plant floor PLCs to enterprise *Servers*. These *Servers* are characterized by a broad scope of size, performance, execution platforms and functional capabilities. Therefore, OPC UA defines a comprehensive set of capabilities, and *Servers* may implement a subset of these capabilities. To promote interoperability, OPC UA defines standard subsets, referred to as *Profiles*, to which *Servers* may claim conformance. *Clients* can then discover the *Profiles* of a *Server*, and tailor their interactions with that *Server* based on the *Profiles*. *Profiles* are defined in [UA Part 7].

The OPC UA specifications are layered to isolate the core design from the underlying computing technology and network transport. This allows OPC UA to be mapped to future technologies as necessary, without negating the basic design. Mappings and data encodings are described in [UA Part 6]. Two data encodings are defined in this part:

- XML/text

- UA Binary

In addition, two transport mappings are defined in this part:

- TCP
- SOAP Web services over HTTP

Clients and *Servers* that support multiple transports and encodings will allow the end users to make decisions about tradeoffs between performance and XML Web service compatibility at the time of deployment, rather than having these tradeoffs determined by the OPC vendor at the time of product definition.

OPC UA is designed as the migration path for OPC clients and servers that are based on Microsoft COM technology. Care has been taken in the design of OPC-UA so that existing data exposed by OPC COM servers (DA, HDA and A&E) can easily be mapped and exposed via OPC UA. Vendors may choose to migrate their products natively to OPC UA or use external wrappers to convert from OPC COM to OPC UA and vice-versa. Each of the previous OPC specifications defined its own address space model and its own set of *Services*. OPC UA unifies the previous models into a single integrated address space with a single set of *Services*.

5.3 Integrated models and services

5.3.1 Security model

5.3.1.1 General

OPC UA security is concerned with the authentication of *Clients* and *Servers*, the authentication of users, the integrity and confidentiality of their communications, and the verifiability of claims of functionality. It does not specify the circumstances under which various security mechanisms are required. That specification is crucial, but it is made by the designers of the system at a given site and may be specified by other standards.

Rather, OPC UA provides a security model, defined in [UA Part 2], in which security measures can be selected and configured to meet the security needs of a given installation. This model includes standard security mechanisms and parameters. In some cases, the mechanism for exchanging security parameters is defined, but the way that applications use these parameters is not. This framework also defines a minimum set of security *Profiles* that all UA *Servers* must support, even though they may not be used in all installations. Security *Profiles* are defined in [UA Part 7].

5.3.1.2 *Session* establishment

Application level security relies on a secure communication channel that is active for the duration of the application *Session* and ensures the integrity of all *Messages* that are exchanged. This means users need to be authenticated only once, when the application *Session* is established. The mechanisms for establishing secure communication channels and application *Sessions* are described in [UA Part 4] and [UA Part 6].

When a *Session* is established, the *Client* and *Server* applications negotiate a secure communications channel and exchange software *Certificates* that identify the *Client* and *Server* and the capabilities that they provide. OPC Foundation-generated software *Certificates* indicate the OPC UA *Profiles* that the applications implement and the OPC UA certification level reached for each *Profile*. The details of each *Profile* and the *Certificates* are specified in [UA Part 7]. *Certificates* issued by other organizations may also be exchanged during *Session* establishment.

The *Server* further authenticates the user and authorizes subsequent requests to access *Objects* in the *Server*. Authorization mechanisms, such as access control lists, are not specified by the OPC UA specification. They are application- or system-specific.

5.3.1.3 Auditing

User level security includes support for security audit trails, with traceability between *Client* and *Server* audit logs. If a security-related problem is detected at the *Server*, the associated *Client* audit log entry can be located and examined. OPC UA also provides the capability for *Servers* to generate *Event Notifications* that report auditable *Events* to *Clients* capable of processing and logging them. OPC UA defines standard security audit parameters that can be included in audit log entries and in audit *Event Notifications*. [UA Part 5] defines the data types for these parameters. Not all *Servers* and *Clients* provide all of the auditing features. *Profiles,* found in [UA Part 7], indicate which features are supported.

5.3.1.4 Transport security

OPC UA security complements the security infrastructure provided by most web service capable platforms.

Transport level security can be used to encrypt and sign *Messages*. Encryption and signatures protect against disclosure of information and protect the integrity of *Messages*. Encryption capabilities are provided by the underlying communications technology used to exchange *Messages* between OPC UA applications. [UA Part 7] defines the encryption and signature algorithms to be used for a given *Profile*.

5.3.2 Integrated *AddressSpace* model

The set of *Objects* and related information that the OPC UA *Server* makes available to *Clients* is referred to as its *AddressSpace*. The OPC UA *AddressSpace* represents its contents as a set of *Node*s connected by *References*.

Primitive characteristics of *Node*s are described by OPC-defined *Attributes*. *Attributes* are the only elements of a *Server* that have data values. Data types that define attribute values may be simple or complex.

*Node*s in the *AddressSpace* are typed according to their use and their meaning. *NodeClasses* define the metadata for the OPC UA *AddressSpace*. [UA Part 3] defines the OPC UA *NodeClasses*.

The *Base NodeClass* defines *Attributes* common to all *Node*s, allowing identification, classification and naming. Each *NodeClass* inherits these *Attributes* and may additionally define its own *Attributes*.

To promote interoperability of *Clients* and *Servers*, the OPC UA *AddressSpace* is structured hierarchically with the top levels standardized for all *Servers*. Although *Node*s in the *AddressSpace* are typically accessible via the hierarchy, they may have *References* to each other, allowing the *AddressSpace* to represent an interrelated network of *Node*s. The model of the *AddressSpace* is defined in [UA Part 3].

OPC UA *Servers* may subset the *AddressSpace* into *Views* to simplify *Client* access. Clause 6.3.3.3 describes *AddressSpace Views* in more detail.

5.3.3 Integrated object model

The OPC UA Object Model provides a consistent, integrated set of *NodeClasses* for representing *Objects* in the *AddressSpace*. This model represents *Objects* in terms of their *Variables, Events* and *Methods*, and their relationships with other *Objects*. [UA Part 3] describes this model.

The OPC UA object model allows *Servers* to provide type definitions for *Objects* and their components. Type definitions may be subclassed. They also may be standardized or they may be system-specific. *ObjectTypes* may be defined by the OPC Foundation, other standards organizations, vendors or end-users.

This model allows data, *Alarms* and *Events*, and their history to be integrated into a single OPC UA *Server*. For example, OPC UA *Servers* are able to represent a temperature transmitter as an *Object* that is composed of a temperature value, a set of alarm parameters, and a corresponding set of alarm limits.

5.3.4 Integrated services

The interface between OPC UA *Clients* and *Servers* is defined as a set of *Services.* These *Services* are organized into logical groupings called *Service Sets*. *Service Sets* are discussed in Clause 7 and specified in [UA Part 4].

OPC UA *Services* provide two capabilities to *Clients*. They allow *Clients* to issue requests to *Servers* and receive responses from them. They also allow *Clients* to subscribe to *Servers* for *Notifications*. *Notifications* are used by the *Server* to report occurrences such as *Alarms*, data value changes, *Events*, and *Program* execution results.

OPC UA *Messages* may be encoded as XML text or in binary format for efficiency purposes. They may be transferred using multiple underlying transports, for example TCP or web services over HTTP. *Servers* may provide different encodings and transports as defined by [UA Part 7].

5.4 Sessions

OPC UA requires a stateful model. The state information is maintained inside an application *Session*. Examples of state-information are *Subscriptions*, user credentials and continuation points for operations that span multiple requests.

Sessions are defined as logical connections between *Clients* and *Servers*. *Servers* may limit the number of concurrent *Sessions* based on resource availability, licensing restrictions, or other constraints. Each *Session* is independent of the underlying communications protocols. Failures of these protocols do not automatically cause the *Session* to terminate. *Sessions* terminate based on *Client* or *Server* request, or based on inactivity of the *Client*. The inactivity time interval is negotiated during *Session* establishment.

5.5 Redundancy

The design of OPC UA ensures that vendors can create redundant *Clients* and redundant *Servers* in a consistent manner. Redundancy may be used for high availability, fault tolerance and load balancing. The details for redundancy are found in [UA Part 4]. Only some *Profiles* [UA Part 7] will require redundancy support, but not the base *Profile*.

6 Systems concepts

6.1 Overview

The OPC UA systems architecture models OPC UA *Clients* and *Servers* as interacting partners. Each system may contain multiple *Clients* and *Servers*. Each *Client* may interact concurrently with one or more *Servers*, and each *Server* may interact concurrently with one or more *Clients*. An application may combine *Server* and *Client* components to allow interaction with other *Servers* and *Clients* as described in Clause 6.3.6.

OPC UA *Clients* and *Servers* are described in the clauses that follow. Figure 3 illustrates the architecture that includes a combined *Server* and *Client*.

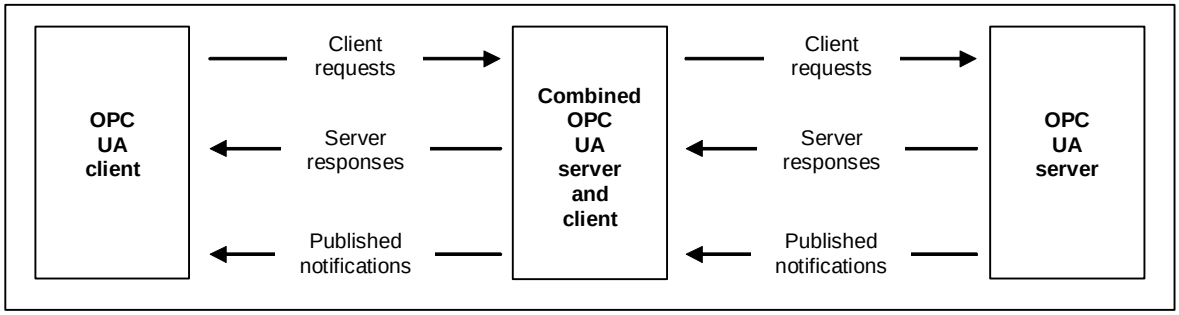

Figure 3 – OPC UA System Architecture

6.2 OPC UA *Clients*

The OPC UA *Client* architecture models the *Client* endpoint of client/server interactions. Figure 4 illustrates the major elements of a typical OPC UA *Client* and how they relate to each other.

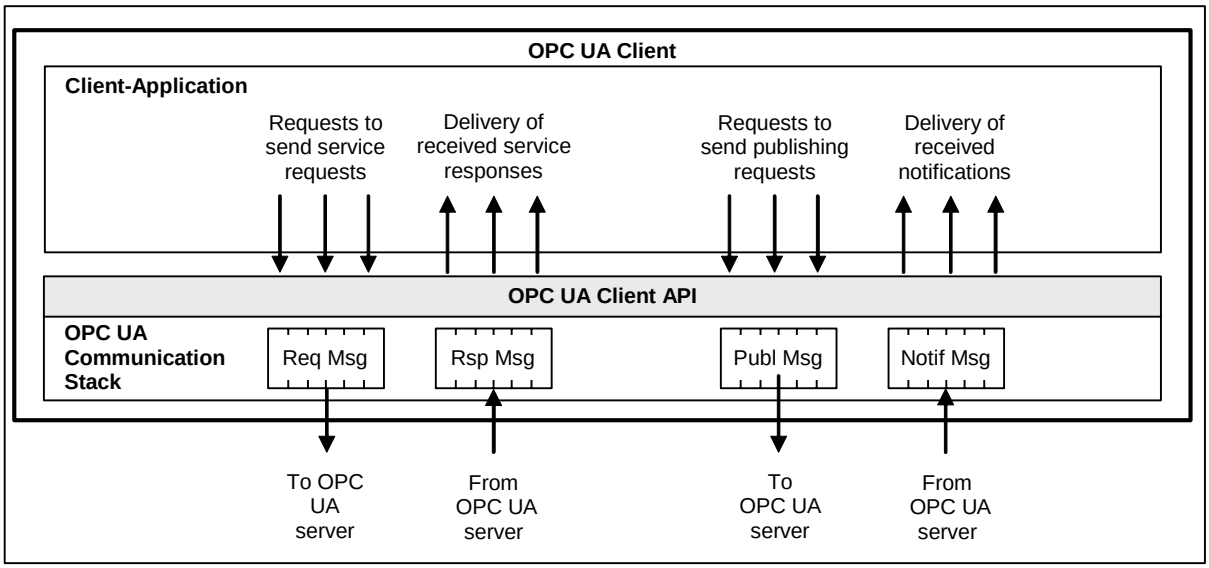

Figure 4 – OPC UA *Client* Architecture

The *Client* Application is the code that implements the function of the *Client*. It uses the OPC UA *Client* API to send and receive OPC UA *Service* requests and responses to the OPC UA *Server*. The *Services* defined for OPC UA are described in Clause 7, and specified in [UA Part 4].

Note that the “OPC UA *Client* API” is an internal interface that isolates the *Client* application code from an OPC UA Communication Stack. The OPC UA Communication Stack converts OPC UA *Client* API calls into *Messages* and sends them through the underlying communications entity to the *Server* at the request of the *Client* application. The OPC UA Communication Stack also receives

response and *Notification Messages* from the underlying communications entity and delivers them to the *Client* application through the OPC UA *Client* API.

6.3 OPC UA *Servers*

The OPC UA *Server* architecture models the *Server* endpoint of client/server interactions. Figure 5 illustrates the major elements of the OPC UA *Server* and how they relate to each other.

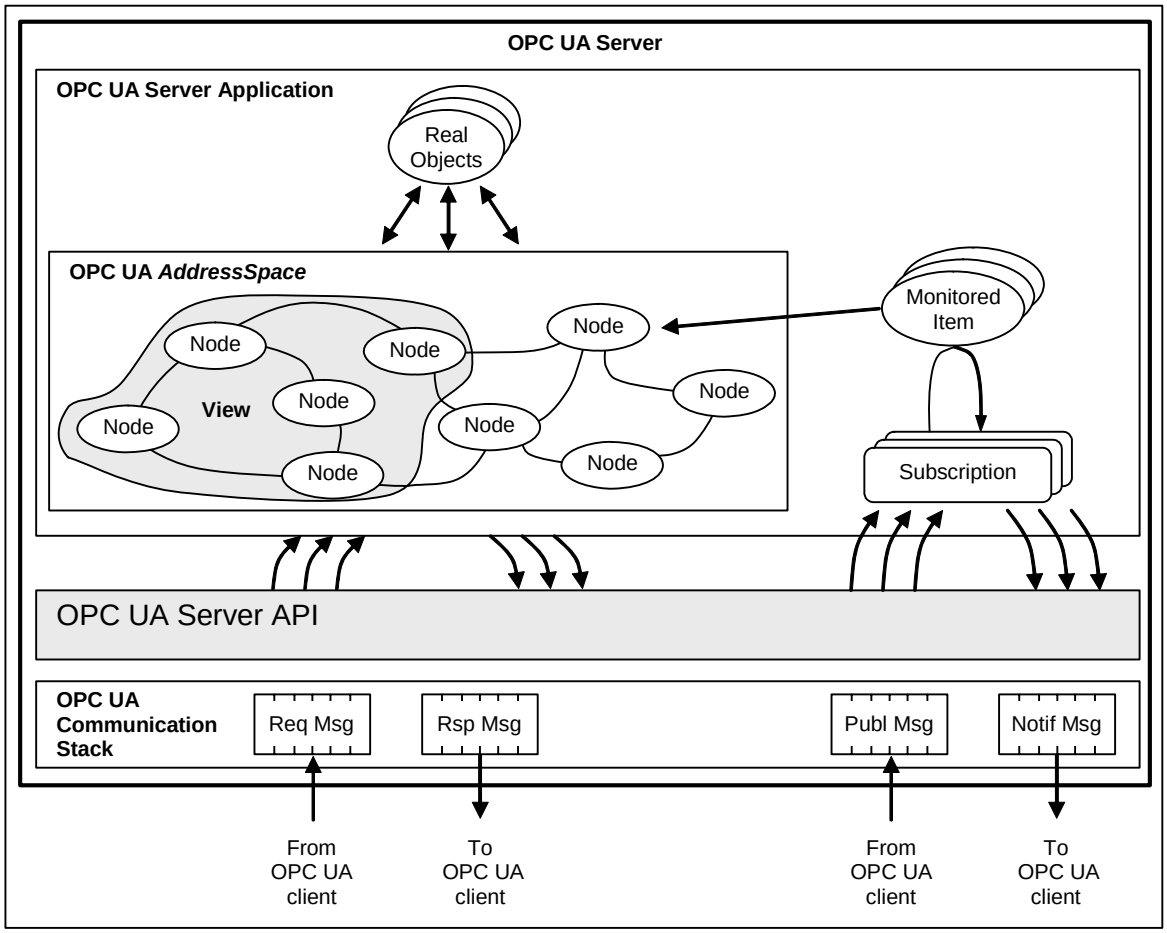

Figure 5 – OPC UA Server Architecture

6.3.1 Real objects

Real objects are physical or software objects that are accessible by the OPC UA *Server* application or that it maintains internally. Examples include physical devices and diagnostics counters.

6.3.2 OPC UA *Server* application

The OPC UA *Server* application is the code that implements the function of the *Server*. It uses the OPC UA *Server* API to send and receive OPC UA *Messages* from OPC UA *Clients*. Note that the "OPC UA *Server* API" is an internal interface that isolates the *Server* application code from an OPC UA Communication Stack. This may be a standard implementation provided by the OPC Foundation or it may be a vendor-specific implementation.

6.3.3 OPC UA *AddressSpace*

6.3.3.1 *AddressSpace Nodes*

The *AddressSpace* is modelled as a set of *Node*s accessible by *Clients* using OPC UA *Services* (interfaces and methods). *Node*s in the *AddressSpace* are used to represent real objects, their definitions and their *References* to each other.

6.3.3.2 *AddressSpace* organization

[UA Part 3] contains the details of the meta model “building blocks” used to create an *AddressSpace* out of interconnected *Nodes* in a standard, consistent manner. *Servers* are free to organize their *Node*s within the *AddressSpace* as they choose. The use of *References* between *Node*s permits *Servers* to organize the *AddressSpace* into hierarchies, a full mesh network of *Node*s, or any possible mix.

[UA Part 5] defines standard OPC UA *Nodes* and *References* and their expected organization in the *AddressSpace*. Some *Profiles* will not require that all of the standard UA *Nodes* be implemented.

6.3.3.3 *AddressSpace Views*

A *View* is a subset of the *AddressSpace*. *Views* are used to restrict the *Node*s that the *Server* makes visible to the *Client*, thus restricting the size of the *AddressSpace* for the *Service* requests submitted by the *Client*. The default *View* is the entire *AddressSpace*. *Servers* may optionally define other *Views*. *Views* hide some of the *Node*s or *References* in the *AddressSpace*. *Views* are visible via the *AddressSpace* and *Clients* are able to browse *Views* to determine their structure. *Views* are often hierarchies, which are easier for *Clients* to navigate and represent in a tree.

6.3.3.4 Support for information models

The OPC UA *AddressSpace* supports information models. This support is provided through:

a) *Node References* that allow *Objects* in the *AddressSpace* to be related to each other.

b) *ObjectType Node*s that provide semantic information for real *Objects* (type definitions).

c) *ObjectType Node*s to support subclassing of type definitions.

d) Data type definitions exposed in the *AddressSpace* that allow industry specific data types to be used.

e) OPC UA companion standards that permit industry groups to define how their specific information models are to be represented in OPC UA *Server AddressSpace*s.

6.3.4 Publisher/subscriber entities

6.3.4.1 MonitoredItems

MonitoredItems are entities in the *Server* created by the *Client* that monitor *AddressSpace Node*s and their real-world counterparts. When they detect a data change or an event/alarm occurrence, they generate a *Notification* that is transferred to the *Client* by a *Subscription*.

6.3.4.2 Subscriptions

A *Subscription* is an endpoint in the *Server* that publishes *Notifications* to *Clients*. *Clients* control the rate at which publishing occurs by sending Publish *Messages*.

6.3.5 OPC UA *Service* Interface

6.3.5.1 General

The *Services* defined for OPC UA are described in Clause 7, and specified in [UA Part 4].

6.3.5.2 Request/response *Services*

Request/response *Services* are *Services* invoked by the *Client* through the OPC UA *Service* Interface to perform a specific task on one or more *Node*s in the *AddressSpace* and to return a response.

6.3.5.3 Publisher *Services*

Publisher *Services* are *Services* invoked through the OPC UA *Service* Interface for the purpose of periodically sending *Notifications* to *Clients*. Notifications include *Events*, *Alarms*, data changes and *Program* outputs.

6.3.6 *Server* to *Server* interactions

Server to *Server* interactions are interactions in which one *Server* acts as a *Client* of another *Server*. *Server* to *Server* interactions allow for the development of servers that:

a) exchange information with each other on a peer-to-peer basis, this could include redundancy or remote *Servers* that are used for maintaining system wide type definitions,

b) are chained in a layered architecture of *Servers* to provide:

   1) aggregation of data from lower-layer *Servers*,

   2) higher-layer data constructs to *Clients*, and

   3) concentrator interfaces to *Clients* for single points of access to multiple underlying *Servers*.

Figure 6 illustrates interactions between *Servers*.

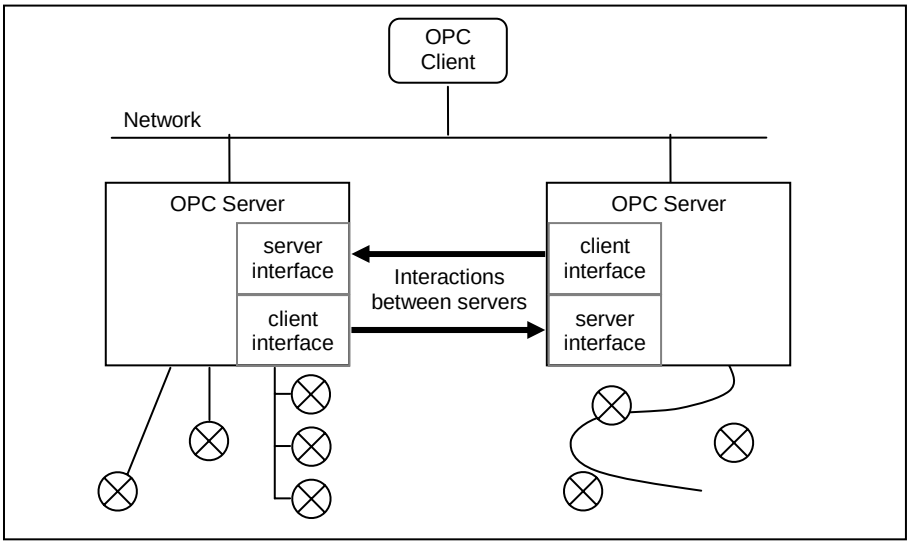

Figure 6 – Interactions between *Servers*

Figure 7 extends the previous example and illustrates the chaining of OPC UA *Servers* together for vertical access to data in an enterprise.

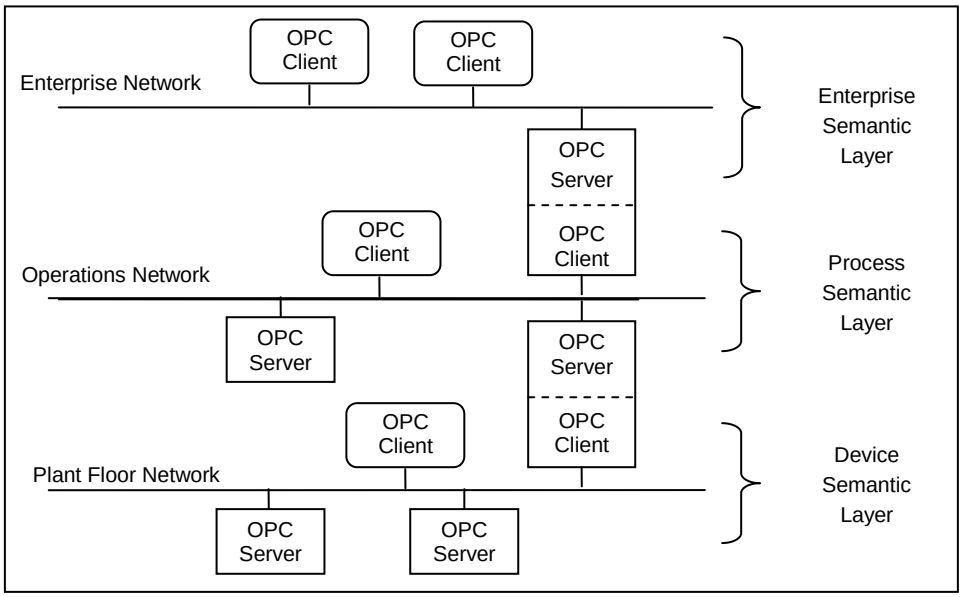

Figure 7 – Chained *Server* Example

7 Service Sets

7.1 General

OPC UA *Services* are divided into *Service Sets*, each defining a logical grouping of *Services* used to access a particular aspect of the *Server*. The *Service Sets* are described below. The *Service Sets* and their *Services* are specified in [UA Part 4]. Whether or not a *Server* supports a *Service Set*, or a specific *Service* within a *Service Set* is defined by its *Profile*. *Profiles* are described in [UA Part 7].

7.2 *SecureChannel* Service Set

This *Service Set* defines *Services* used to discover the security configuration of a *Server* and to open a communication channel that ensures the confidentiality and integrity of all *Messages* exchanged with the *Server*. The base concepts for UA security are defined in [UA Part 2].

The *SecureChannel Services* are unlike other *Services* because they are typically not implemented by the *UA application* directly. Instead, they are provided by the communication stack that the *UA application* is built on. For example, a UA *Server* may be built on a SOAP stack that allows applications to establish a *SecureChannel* using the WS-SecureConversation specification. In these cases, the *UA application* simply needs to verify that a WS-SecureConversation is active whenever it receives a *Message*. [UA Part 6] describes how the *SecureChannel Services* are implemented.

A *SecureChannel* is a long-running logical connection between a single *Client* and a single *Server*. This channel maintains a set of keys that are known only to the *Client* and *Server* and that are used to authenticate and encrypt *Messages* sent across the network. The *SecureChannel Services* allow the *Client* and *Server* to securely negotiate the keys to use.

The exact algorithms used to authenticate and encrypt *Messages* are described in the security policies for a *Server*. A *Client* must select one of the security policies supported by the *Server* when it creates a *SecureChannel*.

When a *Client* and *Server* are communicating via a *SecureChannel* they must verify that all incoming *Messages* have been signed and/or encrypted according to the security policy. A UA

application must not process any *Message* that does not conform to the security policy for the channel.

A *SecureChannel* is separate from the *UA Application Session*; however, a single *UA Application Session* may only be accessed via a single *SecureChannel*. This implies that the *UA application* must be able to determine what *SecureChannel* is associated with each *Message*. A communication stack that provides a *SecureChannel* mechanism but that does not allow the application to know what *SecureChannel* was used for a given *Message* cannot be used to implement the *SecureChannel Service Set*.

The relationship between the *UA Application Session* and the *SecureChannel* is illustrated in Figure 8. The UA applications use the communication stack to exchange *Messages*. First, the *SecureChannel Services* are used to establish a *SecureChannel* between the two communication stacks, allowing them to exchange *Messages* in a secure way. Second, the UA applications use the *Session Service Set* to establish a *UA application Session*.

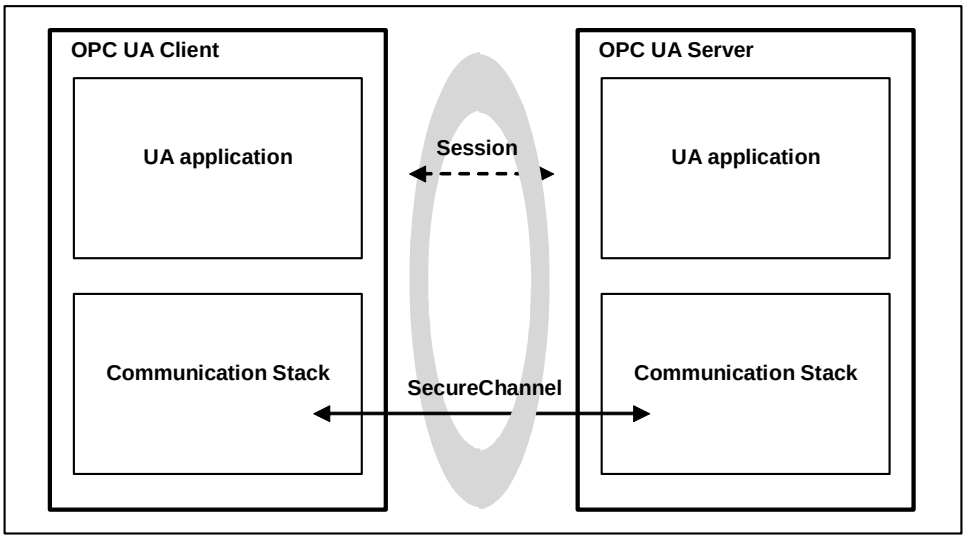

Figure 8 – *SecureChannel* and *Session Services*

When a *Client* establishes a *SecureChannel* it may provide a user identity. This user identity may be different from the user identity provided when the *Client* opens the UA *Application Session*.

7.3 *Session Service Set*

This *Service Set* defines *Services* used to establish an application-layer connection in the context of a *Session* on behalf of a specific user.

7.4 *NodeManagement Service Set*

The *NodeManagement Service Set* allows *Clients* to add, modify, and delete *Node*s in the *AddressSpace*. These *Services* provide a standard interface for the configuration of *Servers*.

7.5 View Service Set

Views are publicly defined, *Server*-created subsets of the *AddressSpace*. The entire *AddressSpace* is the default *View*, and therefore, the *View Services* are capable of operating on the entire *AddressSpace*. Future versions of this specification may also define *Services* to create *Client* defined *Views*.

The *View Service Set* allows *Clients* to discover *Node*s in a *View* by browsing. Browsing allows *Clients* to navigate up and down the hierarchy, or to follow *References* between *Node*s contained in the *View*. In this manner, browsing also allows *Clients* to discover the structure of the *View*.

7.6 Attribute Service Set

The *Attribute Service Set* is used to read and write *Attribute* values. *Attributes* are primitive characteristics of *Node*s that are defined by OPC UA. They may not be defined by *Clients* or *Servers*. *Attributes* are the only elements in the *AddressSpace* permitted to have data values. A special *Attribute*, the *Value Attribute* is used to define the value of *Variables*.

7.7 Method Service Set

Methods represent the function calls of *Objects*. They are defined in [UA Part 3]. *Methods* are invoked and return after completion, whether successful or unsuccessful. Execution times for *Methods* may vary, depending on the function they are performing.

The *Method Service Set* defines the means to invoke *Methods*. A *Method* must be a component of an *Object*. Discovery is provided through the browse and query *Services*. *Clients* discover the *Methods* supported by a *Server* by browsing for the owning *Objects* that identify their supported *Methods*.

Because *Methods* may control some aspect of plant operations, method invocation may depend on environmental or other conditions. This may be especially true when attempting to re-invoke a *Method* immediately after it has completed execution. Conditions that are required to invoke the *Method* may not yet have returned to the state that permits the *Method* to start again. In addition, some *Methods* may be capable of supporting concurrent invocations, while others may have a single invocation executing at a given time.

7.8 *MonitoredItem Service Set*

The *MonitoredItem Service Set* is used by the *Client* to create and maintain *MonitoredItems*. *MonitoredItems* monitor *Variables*, *Attributes* and *EventNotifiers*. They generate *Notifications* when they detect certain conditions. They monitor *Variables* for a change in value, status or timestamp; *Attributes* for a change in value; and *EventNotifiers* for newly generated *Alarm* and *Event* reports.

Each *MonitoredItem* identifies the item to monitor and the *Subscription* to use to periodically publish *Notifications* to the *Client* (see Clause 7.9). Each *MonitoredItem* also specifies the rate at which the item is to be monitored (sampled) and, for *Variables* and *EventNotifiers*, the filter criteria used to determine when a *Notification* is to be generated. Filter criteria for *Attributes* are specified by their *Attribute* definitions in [UA Part 4].

The sample rate defined for a *MonitoredItem* may be faster than the publishing rate of the *Subscription*. For this reason, the *MonitoredItem* may be configured to either queue all *Notifications* or to queue only the latest *Notification* for transfer by the *Subscription*. In this latter case, the queue size is one.

MonitoredItem Services also define a monitoring mode. The monitoring mode is configured to disable sampling and reporting, to enable sampling only, or to enable both sampling and reporting. When sampling is enabled, the *Server* samples the item. In addition, each sample is evaluated to determine if a *Notification* should be generated. If so, the *Notification* is queued. If reporting is enabled, the queue is made available to the *Subscription* for transfer.

Finally, *MonitoredItems* can be configured to trigger the reporting of other *MonitoredItems*. In this case, the monitoring mode of the items to report is typically set to sampling only, and when the triggering item generates a *Notification*, any queued *Notifications* of the items to report are made available to the *Subscription* for transfer.

7.9 *Subscription Service Set*

The *Subscription Service Set* is used by the *Client* to create and maintain *Subscriptions*. *Subscriptions* are entities that periodically publish *Notification Messages* for the *MonitoredItem* assigned to them (see Clause 7.7). The *Notification Message* contains a common header followed

by a series of *Notifications*. The format of *Notifications* is specific to the type of item being monitored (i.e. *Variables*, *Attributes*, and *EventNotifiers*).

Once created, the existence of a *Subscription* is independent of the *Client's Session* with the *Server*. This allows one *Client* to create a *Subscription*, and a second, possibly a redundant *Client*, to receive *Notification Messages* from it.

To protect against non-use by *Clients*, *Subscriptions* have a configured lifetime that *Clients* periodically renew. If any *Client* fails to renew the lifetime, the lifetime expires and the S*ubscription* is closed by the *Server*. When a *Subscription* is closed, all *MonitoredItems* assigned to the *Subscription* are deleted.

Subscriptions include features that support detection and recovery of lost *Messages*. Each *Notification Message* contains a sequence number that allows *Clients* to detect missed *Messages*. When there are no *Notifications* to send within the keep-alive time interval, the *Server* sends a keep-alive *Message* that contains the sequence number of the last *Notification Message* sent. If a *Client* fails to receive a *Message* after the keep-alive interval has expired, or if it determines that it has missed a *Message*, it can request the *Server* to resend one or more *Messages*.

7.10 *Query Service Set*

The *Query Service* Set allows *Clients* to select a subset of the *Node*s in the address space or in a *View* based on some *Client*-provided filter criteria. The *Node*s and their *Attributes* selected by the *Query Service* are called a result set. The result set may be a "flat" set of *Nodes*, or it may preserve the structure by returning *References* connecting the *Nodes* in the result set.

Servers may find it difficult to process queries that require access to runtime data, such as device data, that involves resource intensive operations or significant delays. In these cases, the *Server* may find it necessary to reject the query.