Network Exchange Protocol

Version 1.1

September 17, 2003

Task Order No.: T0002AJM038

Contract No.: GS00T99ALD0203

Abstract

The Network Exchange Protocol V1.1 defines the set of rules intended to govern the generation and use of valid service requests and responses on the Environmental Information Exchange Network (Exchange Network). This Protocol document is intended for use by Node implementers to embed data content standards (defined in Schemas) in service requests and responses. The Protocol described in this document can also be used to confirm or establish the validity of Network service requests and responses.

Amendment Record

Version	Date	Amended By	Nature of Change
Version 1.1a	July 7, 2006	K. Rakouskas	Added Attachment A – Network Operations Board Decision Memorandum 2005-03
Version 1.1	September 17, 2003	A. Reisser	The two parameters - rowId and maxRows are no longer optional.

Table of Contents

Table of Tables

Table of Figures

Foreword

The Network Exchange Protocol V1.0 (Protocol) and the Network Node Functional Specification V1.0 (Specification) define the conversation between and the behavior of Nodes on the Environmental Information Exchange Network (Exchange Network). The Network Steering Board (NSB) expects the Protocol and Specification to have a shelf life of between 12-24 months. As a result, the documents are forward-looking. They define and describe certain functionalities that will not immediately be utilized but are expected to become paramount as the Exchange Network evolves during its initial implementation. For example, the documents discuss and describe UDDI and other Registries as integral parts of the Network. Their use is implicit in the Protocol and Specification, but currently no official registries exist but they do merit discussion in these documents as it is expected that they will exist in the next 12-24 months.

These documents, in their first generation, were/are designed to support relatively simple state and EPA dataflows. They do so by proposing a small number of primitive Network Web services which Network Partners group into larger (but still simple) transactions to flow data. Most of these transactions are now conducted manually through the use of terminal/host clients, email, ftp, http uploads or diskettes. These Web services are:

- Authenticate
- NodePing
- GetServices
- GetStatus
- Notify
- Download
- Submit
- Solicit
- Query

As indicated by the "Authenticate" service, the Protocol and Specification present a decentralized approach for authentication. Each Network Partner is responsible for authenticating users of their Nodes. While allowing optimum flexibility and ultimate control of authentication at the level of the Network Partner, decentralizing authentication could place a resource burden on future Network Partners. The USEPA as part of their Central Data Exchange (CDX) have created the Network Authorization and Authentication Service (NAAS). Any Network Partner can use this service to authenticate users. An additional Web service "Validate," is required, to use the NAAS. The use of the NAAS is described in a separate document, the Network Security Guidelines and Recommendations V1.0 found on the Exchange Network Website. It is expected that in the next 12-24 months, authorization service will be made available at the NAAS. The "Authenticate" service (the process of determining the identity of a subject - not just limited to users; it could, and often should, apply to machines and messages in a secure environment) is nebulous with respect to Nodes or clients. That is, any Node or client can use the "Authenticate" service to obtain authentication. As a result, all potential data exchanges are supported.

As in any software project, these documents represent a series of design decisions and compromises. In their entirety, the Protocol and Specification will strike some implementers as overly complex, and others (or maybe some of the same) as rudimentary. While these documents, created as part of a pilot project, went through several iterations, and represent the most current Network knowledge, the NSB acknowledges that these documents will need updates for several possible reasons including advances in technology.

Critical note to Node implementers:

A WSDL file accompanies the Protocol and Specification. The WSDL file is machine-readable and is the canonical description of the Protocol and Specification. Node implementers should use the WSDL file(s) as the starting point for their Node and client development. Each Node will have to customize the generic WSDL file for their Node. The ability to generate code from the WSDL file is an essential feature of most SOAP toolkits.

Acknowledgements

This document has been developed through the support and analytic contributions of a number of individuals and programs within EPA, ECOS, and several state agencies. These individuals offered valuable insight, lessons learned from their work on this and prior Node workgroups, and hard work in creating the Node V1.0 Specification and Protocol.

State Participants

Dennis Burling (State CoChair), Nebraska Department of Environmental Quality

David Blocher, Maine Department of Environmental Protection

Harry Boswell, Mississippi Department of Environmental Quality

Dan Burleigh, New Hampshire Department of Environmental Services

Frank Catanese, New Hampshire Department of Environmental Services

Ken Elliott, Utah Department of Environmental Quality

Dave Ellis, Maine Department of Environmental Protection

Renee Martinez, New Mexico Environment Department

Tom McMichael, New Mexico Environment Department

Melanie Morris, Mississippi Department of Environmental Quality

Dennis Murphy, Delaware Department of Natural Resources and Environmental Control

Brent Pathakis, Utah Department of Environmental Quality

Brian Shows, Mississippi Department of Environmental Quality

Chris Simmers, New Hampshire Department of Environmental Services

Michael Townshend, Delaware Department of Natural Resources and Environmental Control

Robert Williams, Maine Department of Environmental Protection

Karen Knox, Maine Department of Environmental Protection

EPA Participants

Connie Dwyer (EPA CoChair), Office of Environmental Information

Chris Clark, Office of Environmental Information

Patrick Garvey, EPA NSB Executive Staff

Environmental Council of States

Molly O’Neill, ECOS NSB Executive Staff

Support Contractors

Dave Becker, Computer Sciences Corporation

Tom Potter, Computer Sciences Corporation

Glenn Tamkin, Computer Sciences Corporation

Yunhao Zhang, Computer Sciences Corporation

Andrea Reisser, Concurrent Technologies Corporation

Kochukoshy Cheruvettolil, Ross & Associates Environmental Consulting, Ltd.

Louis Sweeny, Ross & Associates Environmental Consulting, Ltd.

Rob Willis, Ross & Associates Environmental Consulting, Ltd.

State Contractors/Consultants

Tony Pruitt, Ciber Federal Solutions

Steven Wu, enfoTech & Consulting Inc.

Chris McHenry, Integro

Calvin Lee, Oracle

Brad Loveland, Venturi Technology Partners

Brett Stein, XAware Inc.

1.0 Introduction

The Network Exchange Protocol Version 1.0 (V1.0) is a lightweight Protocol for the exchange of structured data, unstructured data, and relational data among Network Nodes across a wide area of Networks. The Protocol defines a framework where data exchanges can take place independent of hardware/software platforms, development tools, and programming languages used.

1.1 Terminology

Term	Definition/Clarification
CSM	Central Security Management
DBMS	Database Management System
DTD	Data Type Definition defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements, (i.e., where each tag is allowed, and which tags can appear within other tags). A DTD is one type of DET.
DET	Data exchange templates identify types of information required or allowable for a particular type of data set according to predefined standards. DETs are empty and contain no data. They simply define the format data must take prior to exchange.
DIME	Direct Internet Message Encapsulation
EPA	Environmental Protection Agency
Exchange Network	Environmental Information Exchange Network
FCD	Flow Configuration Document
HTTP	Hypertext Transfer Protocol
NAAS	Network Authentication and Authorization Services. This is a set of centralized security services shared by all Network Nodes.
QA	Quality Assurance
RBAC	Role-Based Access Control
RPC	Remote Procedure Call
Requester	A Node that initiates SOAP request messages.
SAML	Security Assertion Markup Language
Service Provider	A Node that accepts SOAP messages and executes methods defined by this Protocol.
SMTP	Simple Mail Transport Protocol
SOAP	Simple Object Access Protocol

Term	Definition/Clarification
SQL	Structured Query Language
SSL	Secure Sockets Layer
SSO	Single Sign-on
Target Node	The ultimate destination of a dataflow, a target Node may or may not implement the Network Exchange Protocol V1.0.
tModel	tModel, or Technical Model, is used in UDDI to represent unique concepts or constructs. They provide a structure that allows re-use and, thus, standardization within a software framework. Interfaces defined by the Network Exchange V1.0 Protocol will be registered as tModels in a private UDDI registry.
TRI	Toxics Release Inventory
TRG	Technical Research Group
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
User Node	A Node that uses Network Exchange Protocol V1.0, but does not provide services, also known as pure client.
W3C	World Wide Web Consortium
WSDL	Web Service Definition Language
XML Schema	XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. A Schema is also a type of DET.

2.0 Background

2.1 Principles, Assumptions and Constraints

Principles are rules or maxims that guide subsequent decisions. Principles consist of a list of criteria involving business direction and good practice to help guide the architecture and design.

Assumptions are givens or expectations that form a basis for decisions, and if proven false may have a major impact on the project. They identify key characteristics of the future that are assumptions for the architecture and design, but are not constraints.

Constraints are restrictions that limit options. They are typically things that must or must not be done in designing an application. They identify key characteristics of the future that are accepted as constraints to architecture and design.

The principles, assumptions and constraints for the Network Exchange Protocol V1.0 are as follows.

2.1.1 Principles

1. The Network Exchange Protocol V1.0 should be kept as simple as possible, even if doing so means it will be unable to meet a small number of identified, but advanced needs. The Node Workgroup should prioritize these advanced needs with a premium on simplicity. The rapidly evolving industry Protocol efforts are expected to address these unmet needs, and the Protocol will be adjusted accordingly in the future.
2. The Network Exchange Protocol V1.0 should formalize the Network use cases and provide detailed information about interfacing with Nodes. The Protocol will be used by both Network Flow designers and Network users and should address the needs of these two (2) primary groups of users.
3. The Network Exchange Protocol V1.0 should address how to design the requests and responses (i.e., the Web services) that Network flows should support. Note that the design of the requests and responses will always be driven first and foremost by the immediate needs of those building the flow. However, flow designers should provide end users with the maximum flexibility for data use by keeping the services simple and generic. Designers are encouraged to not focus solely on services that support machine to machine flows between existing systems, but to supplement and extend these with simple services that could be used to support more interactive (if simple) uses.

2.1.2 Assumptions

1. The Network Exchange Protocol V1.0 will rely on existing (if immature) standards (e.g., ebXML messaging Protocol, SOAP, WSDL and UDDI).
2. Immediate development of the Network Exchange Protocol V1.0 is required because:
   a. Many implementers will begin work on Network flows in the fall of 2003.
   b. Even if the initial Network Exchange Protocol V1.0 is imperfect and incomplete, we are better off as a community doing things similarly and consistently so that migration to more stable standards (when they are available) will be easier.

c. Given the immaturity of these technologies, implementers will be looking for any and all practical guidance available.

3. The Protocol will be used by both Network flow designers and Network users.

2.1.3 Constraints

1. The Network Exchange Protocol V1.0 is expected to have a life of 18-24 months. During this time it is likely that significant maturation will have occurred in the broader industry standards efforts and that these will be available as built-in software components to partners' Node software.
2. The technology upon which the Protocol is based is rapidly evolving and will obsolete some portions of the approaches taken.

2.2 Requirements

These requirements describe the technical and functional capabilities that will be delivered as part of the Network Exchange Protocol V1.0. The Network Exchange Protocol V1.0 shall:

1. Support all critical requirements for Network flows including the ability to support processing instructions/transaction type information, such as:
   - The ability to initiate appropriate Network security (See Section 0, Security).
   - The ability to handle different Network uses (See Section 0, Network Exchange Business Processes).
2. Use HTTP/HTTPS, WSDL, and SOAP, and be as consistent as possible in their application with emerging industry standards.
3. Be compatible with Network Security Levels 1-4 (See Section 0 – Security Levels).
4. Able to be implemented using the most common middleware configurations in use by Node implementers, without a high degree of customization.
5. Be both human and machine readable.
6. Character support identification. All Network transactions will be governed by UTF – 8.
7. Support the following message exchange functions:
   a. Synchronous and Asynchronous communication
   b. Acknowledgement
   c. Time stamping

2.3 Out of Scope

The Network Exchange Protocol V1.0 does not govern the following functionality:

- Defining and handling the common types of "missing," "unavailable," or "inapplicable" data. This is an important function but falls outside the scope of the Network Exchange Protocol V1.0.
- Specification of the format of the message payloads.
- Internationalization. There will not be international language support. The standard is English.

3.0 Network Web Services Architecture

The Network Exchange Protocol V1.0 will be used within the larger context of the Network Web services architecture. A software system's architecture defines the overall structure of the system. It partitions the system into components, allocates responsibilities among those components, defines how the components collaborate, and how control flows through the system.

3.1 A Basic Web Services Architecture

Service Provider – This is the provider of the Web service. The service provider implements the service, publishes its availability, makes it available on the Internet, and processes requests for services.

Service Requester – This is any consumer of the Web service. The service requester discovers an existing Web service, retrieves its description, and then utilizes the Web service by opening a Network connection and sending an Extensible Markup Language (XML) request conforming to its interface description.

Service Registry – This is a logically centralized directory of Web services. The service registry provides a central place where service providers can publish new Web services and service requesters can find existing ones.

The basic components of any Web services architecture are depicted in Figure 1.

The typical order of operations of basic Web services is also depicted in Figure 1. The arrows in the diagram flow from the initiating component and show the direction of the request as detailed below:

1. The service provider develops their service and publishes its availability in the service registry using Universal Description Discharge, and Integration (UDDI).
2. The service requester accesses the service registry (using UDDI) to find the service with which they want to work. They retrieve a pointer (using UDDI) to a description of the service (typically a detailed technical specification of how to interact with the service), and they retrieve the actual address (using UDDI) of the service.
3. The service requester retrieves the service description Web Service Definition Language (WSDL) using the pointer it obtained from the service registry. The service description would be located in a separate repository.
4. The service requester then formulates its service request using the detailed specification of the service description, and sends the request to the service at the address also retrieved from the UDDI registry.

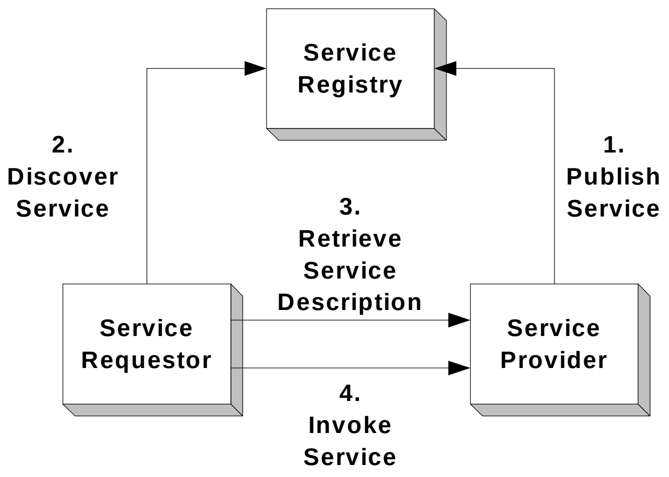


Figure 1 - Basic Components of the Network Web Services Architecture

3.2 Extending the Basic Web Services Architecture for the Network

The basic Web services architecture described above will be extended to implement the Network. This will require additional components and result in a more complex flow of operations.

The components and the flow of operations of the Network Web services architecture is best depicted in the two separate diagrams below. Figure 2 depicts the setup of the Network, while Figure 3 depicts the operation of the Network once it is set up.

3.2.1 Additional Components of the Network

The additional components of the Network Web services architecture depicted in the figures are as follows:

DET Registry - This is a logically centralized directory of Data Exchange Templates ((www.exchangeNetwork.net). DETs are the XML Schemas that describe the various payloads (data files) that may be exchanged across the Network. The DET registry provides a central place where the DET Authority, the Technical Research Group (TRG) can publish new DETs for subsequent discovery.

DET Repository - This is a logically centralized storage location for the DETs (www.exchangenetwork.net). The DET repository provides a central place where the DET Authority, and the TRG can store new DETs for subsequent retrieval.

Flow Configuration Document (FCD) Registry - This is a logically centralized directory of FCD. The FCD defines the business rules and parameters that will be in effect between a given service requester and service provider. The FCD registry provides a central place where Network participants can publish new FCDs. FCDs have traditionally been paper documents signed by the parties to the agreement. However, they can also exist in executable form supplying needed information to help automate business transactions that occur within the scope of the agreement.

Service Description Repository - This is a logically centralized storage location for the Service Descriptions, also called WSDL files. The service description repository provides a central place where the parties to a trading partner agreement can store new service descriptions for subsequent retrieval.

DET Authority -The DET authority is the TRG. It has responsibility for reviewing and approving the DET and administering its availability for other applications to use.

3.2.2 Setup of the Network

Setup of the Network will be an ongoing process as new services are added, and older services are updated or retired. The setup of the Network Web services architecture as depicted in Figure 2 is as follows:

1. The DET authority (the TRG), which is responsible for administering the XML schema definitions for each of the exchange payloads that will be moving across the Network as part of a given flow, defines an official version of the XML schema definition and stores it in the DET repository.
2. The TRG then publishes the official version of the XML schema definition in the DET registry.
3. The service provider develops their service, creates a service description using WSDL, and stores the service description in the service description repository.
4. The service provider then stores the availability of their Web service in the service registry (using UDDI, See Reference 15 – UDDI Version 3.0).
5. The service requester and the service provider publish their FCD in the FCD registry. They also store the parameters associated with the business rules governing their information exchange in a Technical Mode (tModel) in the FCD registry.

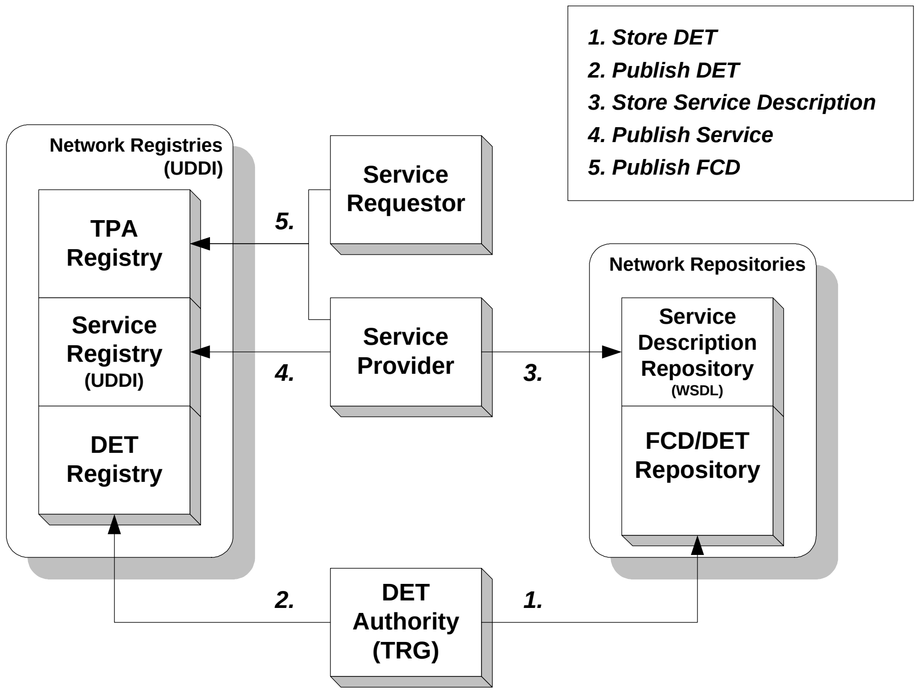


Figure 2 - Setup of the Network

3.2.3 Operation of the Network

The typical order of operations of the Network Web services architecture as depicted in Figure 3 is as follows:

1. The service requester accesses the service registry (using UDDI, See Reference 15 – UDDI Version 3.0) to find the service with which they want to work. They retrieve a pointer (using UDDI) to a description of the service, and the actual address (using UDDI) of the service.
2. The service requester retrieves the service description (WSDL, See reference 16) from the service repository using the pointer it obtained from the service registry.
3. The service requester retrieves the FCD and its business rules from the FCD registry.
4. The service requester formulates its service request using the detailed specification of the service description and the business rules from the FCD. This service request is sent to the service at the address retrieved from the service registry.
5. The service provider retrieves the business rules from the FCD registry, validates the service request, and then performs the requested activity, typically retrieving requested information.
6. The service provider retrieves the payload schema definition from the DET registry and uses it to decode the payload.
7. The service provider validates the payload result and, processes the request and then returns the response to the requester.
8. The service requester retrieves the payload schema definition from the DET registry, validates the response, and uses the information as appropriate for its own purposes.

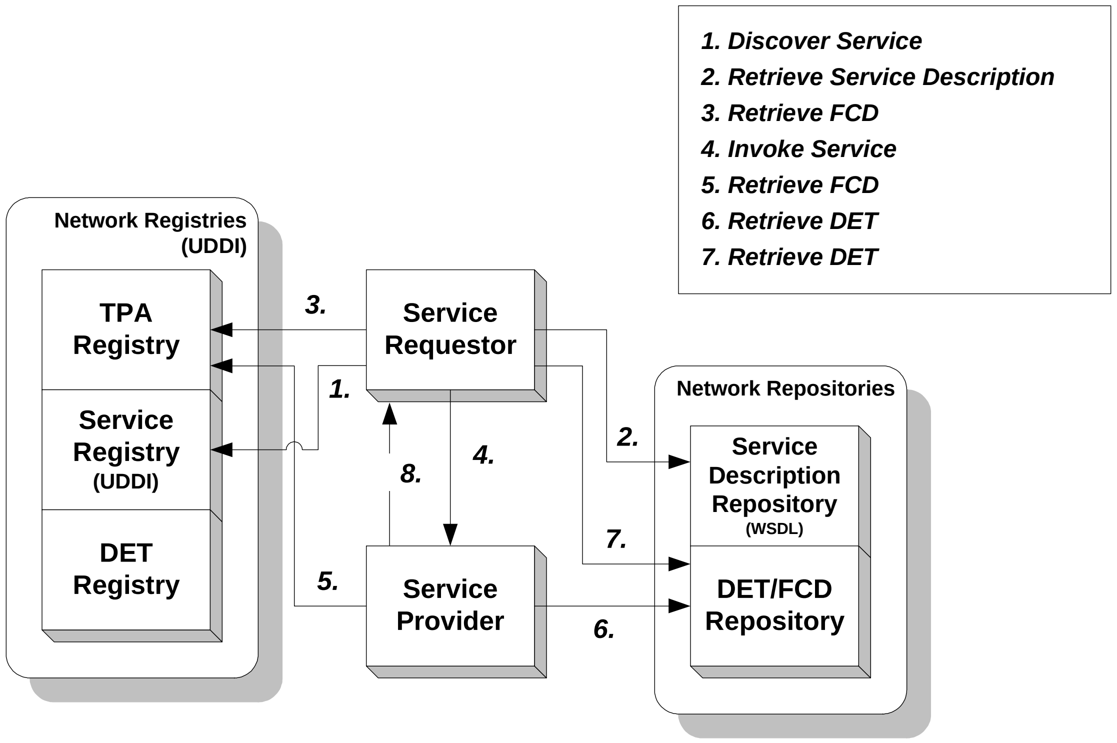


Figure 3 - Operation of the Network

3.3 Network Registries and Repositories

The Network registries and repositories may actually be housed in the same physical location and use the same general access services. However, each of these registries and repositories must be treated as logically separate entities.

In addition, any or all of the three possible Network Registries, as well as the service registry may utilize a "Registrar" service (not pictured in Figure 2). The registrar provides UDDI registration services on behalf of a customer (e.g. a Web service provider). It is responsible for handling additions of entries to the registry and updates and deletes of registered entries in the registry. A registrar can be a Website that provides a human interface to the customer and then employs the API for accessing the registry or the registrar can be totally automated.

3.4 Network Web Services Protocol Stack

The basic Protocol can also be visualized as a stack of several layers of capability with various standards applicable to each layer:

Discovery	UDDI
Description	WSDL
XML Messaging	SOAP, XML
Transport	HTTP/HTTPS
Security	SSL

Each layer is independent from the layers above and below it. Each has its own job that provides greater flexibility allowing the connection of all forms of disparate systems and Network technologies to support distributed processing over the Internet.

3.4.1 Security

This layer insulates the application from unwanted intrusion and unauthorized access. It can employ a number of different security Protocols. However, the approach that must be supported by the Network at this time is Secure Sockets Layer (SSL) plus service level user authentication and authorization (user name and password).

3.4.2 Transport

This layer is responsible for transporting messages between applications. It can also employ a number of different Protocols. However, the transport Protocol that must be supported by the Network at this time is Hypertext Transfer Protocol HTTP/HTTPS / 1.1

3.4.3 XML Messaging

This layer is responsible for encoding messages in a common XML format so that the messages can be understood at either end. The approaches that must be supported by the Network at this time are: a) Simple Object Access Protocol (SOAP) / 1.1 for the encoding of the message structure; and b) XML Schema for the encoding of the message payload.

3.4.4 Service Description

This layer is responsible for describing the interface to a specific Web service. The approach that must be supported by the Network at this time is WSDL / 1.1(WSDL, See Reference 16).

3.4.5 Service Discovery

This layer is responsible for centralizing services into a common registry and providing publishing/finding functionality. The current approach for providing this functionality is UDDI (UDDI, See Reference 15).

3.5 Web Services Standards

At each layer of the Web services Protocol stack there are one or more applicable standards that must be understood and addressed.

3.5.1 Secure Socket Layer (SSL)

SSL is a Protocol originally designed by Netscape to encrypt messages sent across the Internet using HTTP. SSL ensures that no one can easily intercept the messages and read them, thus providing a significant degree of privacy in Internet communications. SSL is a separate layer that sits below HTTP and above TCP and IP. HTTP over SSL has a default port of 443, as opposed to HTTP's default port of 80. This means that many applications will have two (2) default ports, 80 and 443.

SSL is technically proprietary, although just about every browser has implemented it. There is an effort underway to turn SSL into an open standard, something called Transport Layer Security (TLS).

3.5.2 Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) was designed by Tim Berners-Lee at CERN in Europe to make scientific paper (document) communications between computers easy by specifying a set of rules of conversation. It requires the presence of applications, which follow different rules in the conversation and act as either clients or servers. Clients always initiate the contact and start the conversation, while servers can only respond to requests from clients. The client makes a request, the server makes a response, and then the two completely forget about each other resulting in a stateless connection.

3.5.3 Simple Object Access Protocol (SOAP)

SOAP usage has expanded beyond its implementation with Objects. SOAP is an XML-based Protocol for exchanging information between computers. There is a very low level alternative to SOAP called XML Remote Procedure Call (RPC).

IBM, Microsoft, Ariba, and others originally contributed to create SOAP. SOAP 1.1 was submitted to the World Wide Web Consortium (W3C) in May 2000 (See Reference 7). The W3C created an XML Protocol Working Group, which is attempting to develop an official recommendation. This group has released a "Working Draft" of the new SOAP standard, Version 1.2. However, it currently only has the status of "Note". The W3C is also considering a separate "SOAP Messages with Attachments". Both of these approaches – with the payload embedded in the body, and the payload as an attachment – have been encountered in tools used by the members of the Node Beta project, will be supported.

3.5.4 Extensible Markup Language (XML)

XML is a language for writing markup languages. Using XML a user can create a tag-based markup language for the representation of information about almost any topic possible. The structure and content of the markup language is defined either at a high level through a formal specification called a document type definition (DTD), or at a more detailed level typically through an XML Schema (itself specified through XML). An instance of information in the markup language encoded/marked-up according to one of these specifications is called an XML document, and will contain tags identifying the content by a series of elements and attributes associated with the content in the order and format as specified. The formal specifications can be used to automatically validate an XML document using a validating XML parser.

XML is an open standard with Unicode as its standard character set. It is readable by both humans and machines, and is being widely adopted in almost all modern information exchange situations (e.g., it is rapidly replacing EDI for electronic commerce applications). XML has been adopted by the Environmental Protection Agency (EPA) TRG for representation of information that will be flowing across the Network. Separate XML specifications (XML Schemas) have been or are being drafted for dataflows.

XML was the original standard around which the majority of activity of the W3C was formed. It is now an official recommendation of the W3C. It is currently at Version 1.0.

3.5.5 Web Services Description Language (WSDL)

WSDL, See Reference 16, is an XML-based language specification defining how to describe a Web service in computer readable form. For a given Web service, its WSDL file describes four (4) key pieces of data:

1. Interface – information describing all available functions/methods.

2. Data type – information for all message requests and message responses.
3. Binding – information about the transport Protocol to be used.
4. Address – information for locating the specified service.

WSDL represents the contract between the service requester and the service provider. Using WSDL, a client can locate a Web service and invoke any of its available functions. With WSDL aware tools, you can automate this process. There were originally several other proprietary attempts to create a similar specification (IBM's NASSL and Microsoft's SCL). But WSDL is rapidly becoming the de facto standard for carrying out this functionality. WSDL aware SOAP Toolkits have significant advantages in being able to automate this process and save significant resources and time however, support varies widely across the market and a detailed evaluation against the specification requirements is necessary to select a good tool (See Soap Toolkit Selection Guide).

IBM, Microsoft and Ariba among others originally created WSDL. It was submitted to the W3C in March 2001. WSDL is not an official recommendation of the W3C. It currently has no official status. It will probably be placed under the W3C Web Services Activity's Web Services Description Working Group. It currently exists in Version 1.1.

3.5.6 Universal Description, Discovery, and Integration (UDDI)

UDDI (UDDI, See Reference 15) is a technical specification that provides a programmatic way for people to find and use a certain Web service. UDDI is a critical part of the emerging Web services Protocol stack. It enables organizations to both publish and find Web services. Today this function is performed manually in a very ad hoc, hit-and-miss fashion. There are no other potential standards that currently exist in this area. Additionally, UDDI acceptance has been slowed by validation and security problems at the public UDDI registries that result in many useless listings (incorrect links and dead links).

Microsoft, IBM and Ariba originally announced V1.0 of UDDI in September 2000. Since the initial announcement, the UDDI initiative has grown to include nearly 300 companies. In June 2001, the UDDI group announced V2.0. According to the original plan, the UDDI group will release three versions of UDDI, and then turn the specification over to an appropriate standards body.

4.0 Network Message Structure

All Network messages will utilize the basic HTTP request/response structure. Within this basic transport layer structure, all messages will be encoded using SOAP's envelope/header/body structure in which header is optional. Inside the body of the SOAP message, the payload will be encoded using XML (XML Schema). The payload will typically be a simple request, a document response or an error response (called a fault). The response will be an answer to the request. This basic structure is depicted in Figure 4.

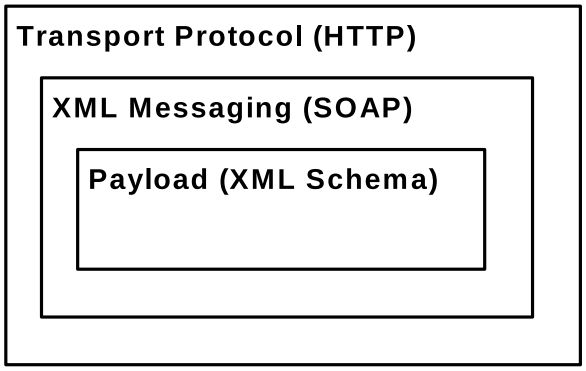


Figure 4 - Network Protocol Message Structure

The three primary components of the message structure that need to be discussed are the transport Protocol, HTTP, the XML messaging Protocol, SOAP, and the payload encoded according to an XML schema. Because SOAP is being used over HTTP, it imposes some constraints on what must or must not be included in the HTTP message structure. Also, because XML payloads are being used in the SOAP messages, the XML is imposing certain constraints on the SOAP message structure.

4.1 HTTP Transport Protocol

The only currently supported transport mechanism approved, as part of the Network Exchange Protocol V1.0 is HTTP/HTTPS.

HTTP is a two-message system of communication. There is a request HTTP structure and a response HTTP structure. All Network messages will utilize the basic HTTP request/response structure. SOAP requests are sent via an HTTP request and SOAP responses are returned within the content of an HTTP response.

HTTP Request – The HTTP request is composed of:

1. A request line which consists of a method, a URL, and the version of HTTP being used.
2. Optional message headers.
3. A blank line (carriage return and line feed, CR + LF).
4. An optional message body.

HTTP Request Example:

```
POST /NodeServer/ HTTP/1.1
User-Agent: Mozilla/4.0
Host: www.epa.gov
```

```
Content-Length: nnnn
SOAPAction: “”

<?xml version-‘1.0’ encoding=’UTF-8’?>
<env:Envelope …
…
</env:Envelope>
```

A normal HTTP method can be one of four possible choices: GET, POST, PUT and DELETE. Currently the SOAP specification only specifies use of the POST method. The URL is the location (directory structure on the Web server) where the Web service can be found. The version of HTTP being used is typically “HTTP/1.1”.

A variety of HTTP message headers can be incorporated in order to specify a wide range of information. This could include specifying what type of information can be accepted by the requesting client, the name and version of the agent/client making the request, the name of the host machine making the request, etc. However, SOAP / 1.1 requires that there always be at least one header present, a “SOAPAction” header. The SOAPAction header must contain either a URI (the name and location of the actual procedure which is being called), or it must be blank. The only real purpose of the SOAPAction header is to allow servers, such as firewalls, to appropriately filter SOAP messages. The SOAPAction header is actually depreciated in the SOAP / 1.2 specification.

The blank line must be present to separate the HTTP request line and headers from the body of the message.

The message body must be encoded in XML according to the SOAP conventions discussed below.

HTTP Response – The HTTP response looks much like the request with some small but significant differences. The HTTP response is composed of:

1. A status line which consists of the version of HTTP being used, a numerical code describing the status of the server response, and a string describing the status.
2. Optional message headers.
3. A blank line (carriage return and line feed, CR + LF).
4. An optional message body.

HTTP Response Example:

```
HTTP/1.1 200 OK
Date: Mon, 26 Jun 2002 14:22:54 GMT
Server: Apache/1.3.20 (Unix)
Connection: close
Content-Type: text/XML
<?xml version-‘1.0’ encoding=’UTF-8’?>
<env:Envelope …
…
```

```
            </env:Envelope>
```

SOAP RPCs masquerade as standard HTTP messages (i.e., operate on top of HTTP). Because HTTP is allowed through most firewalls via a standard port, this enables SOAP RPCs to easily penetrate most firewalls, and invoke their targeted procedures.

Many vendors have implemented HTTP. However, because it is so mature, and is widely accepted and used, it is very unlikely those different HTTP implementations will result in interoperability issues. As a result, a standard Network Exchange Protocol V1.0 HTTP implementation will not be defined.

4.1.1 SMTP as an Additional Transport Mechanism

An additional transport layer Protocol that is being considered for moving SOAP messages is Simple Mail Transport Protocol (SMTP). SMTP is used to move messages, and frequently large quantities of information, from a source to a destination. This is accomplished asynchronously in a fire-and-forget fashion. It is very efficient at moving information one way.

SMTP operates on a different port than HTTP, but this too is frequently enabled through many firewalls. Use of SMTP could potentially result in added overhead in having to navigate the firewall.

While SMTP is being investigated as a SOAP transport Protocol, the SOAP specification is currently only defined for HTTP. However, as the development of the Network Exchange Protocol V1.0 continues, it is important to clearly identify that this is an area of the Network Exchange Protocol V1.0 that would have to be enhanced if additional SOAP transports are needed, such as SMTP.

4.2 SOAP Messaging

All Network transactions must be SOAP messages. SOAP is bound to HTTP. The Network Exchange Protocol V1.0 does not currently support SOAP binding to other transport mechanisms. All Nodes must support SOAP / 1.1 as defined by the W3C. SOAP messages are composed of a mandatory envelope element, an optional header element, a mandatory body element and an optional fault element. All Network payloads are carried in the body of the SOAP message or as an attachment to the envelope. This basic structure is depicted in Figure 5.

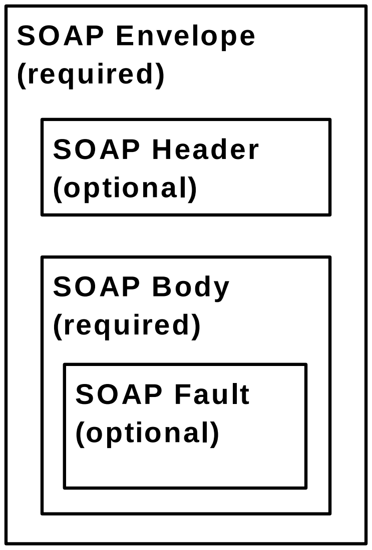


Figure 5 - Network SOAP Message Structure

The Network Exchange Protocol V1.0 does not govern payload issues. However, it is expected that the SOAP XML message structure for all SOAP messages will be validated with the Network SOAP schema located in the Network registry.

It is anticipated that various Web services, and SOAP automation tools, will be used to generate the SOAP message structure automatically from inputs supplied by the user or the application. In these cases, the SOAP message structure will be largely invisible to the users and applications. However, there may still be instances where it will be desirable to generate or inspect the actual SOAP message structure. This section is intended for both those who wish to carry out this type of activity, and for those who need to validate/certify that their Web service requesters and providers are operating properly.

Table 1 contains all required SOAP tags and the details of Network use. Any SOAP message that does not contain all of the tags and/or the use of the required tag is inconsistent with the Network uses defined in Table 1 and is not considered conforming to Network standards. It is therefore violating terms defined in the governing FCD. Furthermore, messages not conforming to the Network Exchange Protocol V1.0will not pass the required Network SOAP validation.

Soap Element	Network Use	Supported Use Cases
Envelope	Specify SOAP version being used	All
Body	Encapsulate message payload (requests, responses, and faults)	All
Fault	Error handling responses	SOAP Error
fault code	Error handling; required within fault element	SOAP Error

Soap Element	Network Use	Supported Use Cases
faultstring	Error handling; required within fault element	SOAP Error
faultactor	Error handling; optional within fault element	SOAP Error

Table 1 - Mandatory Soap Tags

Table 2 contains all SOAP tags that are not required by the Network Exchange Protocol V1.0 but are supported by the Network. Use of these tags is optional and does not constitute a violation of the Network Exchange Protocol V1.0.

Soap Element	Network Use	Supported Use Cases
Header	Additional processing information	Request, notification, solicit, and one-way
Fault detail	Error handling; optional within fault element	SOAP Error

Table 2 - Optional SOAP Tags

Table 3 contains all SOAP tags that are prohibited for Network use. Any instance that uses the WSDL tags contained in Table 3 constitutes a violation of the Network Exchange Protocol V1.0 and non-conformance to the governing FCD. At this time there are no prohibited SOAP tags in the Network Exchange Protocol V1.0.

Soap Element	Network Use	Supported Use Cases
—N/A	N/A	N/A

Table 3 - Prohibited SOAP Tags

Network Service Request Example:

```
POST /cdx/node.wsdl HTTP/1.1
Date: Mon, 26 Jun 2002 14:22:54 GMT
Connection: close
Content-Type: text/XML
Content-Length: xxxx
<?xml version=‘1.0’ encoding=’UTF-8’?>
<env:Envelope …
   xmlns:env=”http://www.w3.org/2001/06/soap-envelope/”>
   <SOAP-ENV:Body
      <exchangenetwork:basicRequest
```

```
                         xmlns:exchangenetwork=”http://www.exchangenetwork.net/sc
hemas/v1.0/node.wsdl/”
                         SOAP-ENV:encodingStyle=
                            ”http://xml.apache.org/xml-soap/literalxml/”>
              …
              </env:Envelope>
```

4.2.1 SOAP Envelope

The **envelope** element is the root element of the SOAP message. The rest of the SOAP message must be contained within the **envelope** start and end tags. The **envelope** element must be prefixed with an indicator of the namespace that defines the SOAP version that is applicable. The version is indicated by the namespace attribute, **xmlns**, included in the envelope element start tag. The namespace prefix could be any valid XML namespace string, but the convention usually adopted is as follows:

```
          <SOAP-ENV:Envelope
             xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”>
```

The namespace name SOAP-ENV is really a symbol for http://schema.xmlsoap/soap/envelope. Although it can be any NCName (An XML Name, minus the ":"), the URL part must be exactly as specified. A different URL represents a different version of SOAP and must cause the VersionMismatch fault (see Section 0 for definition).

4.2.2 SOAP Header

The header element is used to provide information about the message.

4.2.2.1 MustUnderstand Attribute

Must be supported by the Network.

4.2.2.2 Actor Attribute

Not supported by the Network at this time.

4.2.3 SOAP Body

The body element is used to provide information about the message.

4.2.3.1 Encoding

Encoding style governs how a SOAP message is serialized and deserialized. The SOAP 1.1 specification defines only one encoding style, (i.e., SOAP encoding), also known as the Section 5, encoding. The encoding style is identified by the URL: “http://schemas.xmlsoap.org/soap/encoding/”, which points to the SOAP/1.1 encoding schema. An empty encoding style or missing encoding style indicates that no claims are made for the encoding style of contained elements.

The encoding style attribute, according to the SOAP/1.1 specification, can appear on any element in a SOAP message. To simplify message processing, the Network Exchange Protocol V1.0 imposes the following restrictions:

1. The encoding style must not appear on the SOAP envelope element.
2. No other encoding styles are allowed in the child elements of SOAP-ENV:Body or SOAP-ENV:Header under Section 5 encoding, except an empty encodingStyle attribute.

The first rule makes it possible to have different encoding styles for SOAP header and body (which has been adapted by the SOAP/1.2 specification); the second rule prevents a message from having mixed encoding styles in either the body part or the header part, which simplifies message processing. The following example shows a non-section 5 encoding governed by :http://xml.apache.org/xml-soap/literalxml/.

```
      <SOAP-ENV:Body
            <exchangenetwork:basicRequest
               xmlns:exchangenetwork="http://www.exchangenetwork.net/schemas/
v1.0/node.wsdl"
               SOAP-ENV:encodingStyle=
                  "http://xml.apache.org/xml-soap/literalxml/">
                  …
            </exchangenetwork:basicRequest>
         </SOAP-ENV:Body>
```

SOAP Encoding

Used for requests; A superset of XML Schema; Must be specified in the body.

```
      <SOAP-ENV:Body
            <exchangenetwork:basicRequest
               xmlns:exchangenetwork="http://www.exchangenetwork.net/schemas/
v1.0/node.wsdl"
               SOAP-ENV:encodingStyle=
                  "http://schemas.xmlsoap.org/soap/encoding/">
                  …
            </exchangenetwork:basicRequest>
         </SOAP-ENV:Body>
```

Literal Encoding

The literal encoding style allows arbitrary XML elements to be sent in a SOAP message. It has been a common practice to set the encoding style attribute to empty in such a situation.

SOAP messages of literal encoding are often governed by XML schema rather than encoding styles. The following example shows the structure of a literal encoded message:

```
      <SOAP-ENV:Body
            <exchangenetwork:basicRequest
               xmlns:exchangenetwork="http://www.exchangenetwork.net/schemas/
v1.0/node.wsdl"
               SOAP-ENV:encodingStyle=
                  "">
                  …
            </exchangenetwork:basicRequest>
         </SOAP-ENV:Body>
```

Although SOAP does not mandate the position of the encoding style namespace, it should, in general, not be an attribute of the SOAP envelope. This allows the SOAP header and body to have different encoding styles, for instance, an RPC encoded header but document/literal encoded body.

4.2.4 SOAP Fault

4.2.4.1 SOAP Fault Codes

The SOAP/1.1 Protocol defines four fault codes that must be used in all SOAP fault messages. They are referenced in Table 4.

Fault Code	Meaning
VersionMismatch	The SOAP envelope namespace is wrong.
MustUnderstand	A header with mustUnderstand set to 1 could not be processed (understood) by the receiver.
Client	Client message is invalid or could not be processed.
Server	A fault caused by a server-side error

Table 4 - SOAP Fault Code

4.2.4.2 SOAP Fault Detail Codes

All SOAP fault messages must confirm to the SOAP/1.1 specification and use the predefined SOAP fault codes. In addition, all SOAP fault messages must contain a fault detail element, with Network exchange specific error codes and error descriptions, when processing of a SOAP request body fails.

The error codes for the Network Exchange Protocol V1.0 are listed in Table 5.

Error Code	Description
E_UnknownUser	User authentication failed.

Error Code	Description
E_Query	The supplied database logic failed.
E_TransactId	A transaction Id could not be found.
E_UnknownMethod	The requested method is not supported.
E_ServiceUnavailable	The requested service is unavailable.
E_InvalidToken	The securityToken is invalid.
E_AccessDenied	The operation could not be performed due to lack of privilege.
E_TokenExpired	The securityToken has expired.
E_FileNotFound	The requested file could not be located.
E_ValidationFailed	DET validation error.
E_ServerBusy	The service is too busy to handle the request at this time, please try later.
E_RowIdOutofRange	The rowId parameter is out of range.
E_FeatureUnsupported	The requested feature is not supported.
E_VersionMismatch	The request is a different version of the Protocol.
E_InvalidFileName	The name element in the Node Document structure is invalid.
E_InvalidFileType	The type element in the Node Document structure is invalid or not supported.
E_InvalidDataFlow	The dataflow element in a request message is not supported.
E_InvalidParameter	One of the input parameter is invalid.
E_InternalError	An unrecoverable error occurred during processing the request.
E_InvalidSQL	Syntax error in the SQL statement.
E_AuthMethod	The authentication method is not supported.
E_AccessRight	User privilege is insufficient for the operation.
E_InvalidFileName	The name element in the NodeDocument structure is invalid.

Table 5 - Network Exchange Error Code

The message below shows the structure of a SOAP fault message with the fault detail element:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
       <SOAP-ENV:Fault>
           <faultcode>SOAP-ENV:Client</faultcode>
           <faultstring>Access Denied</faultstring>
           <detail>
               <e:faultdetails
xmlns:e="http://www.exchangenetwork.net/schema/v1.0/node.xsd">
                 <errorcode>E_TokenExpired</errorcode>
                 <description>The securityToken has expired.</description>
               </e:faultdetails>
           </detail>
       </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The fault detail element (in bold) indicates that the fault is due to an invalid authentication token, a fault that is specific to this Protocol. The fault detail element must be a qualified element, governed by the namespace URL: http://www.exchangenetwork.net/schema/v1.0/node.xsd.

Note that the value of fault code, *SOAP-ENV:Client*, is also namespace-qualified. Although the namespace prefix is only recommended by the SOAP/1.1 specification, it is a requirement by this specification that all fault code values be namespace-qualified to reduce ambiguity.

4.3 XML Payloads

4.3.1 Payload Location

All Network transactions must be SOAP messages. Specific payloads that are being transferred between trading partners will either be enclosed within the body of the SOAP message or attached to the SOAP message.

4.3.1.1 Embedded Payloads

All payloads that are RPC-style messages must be base64 encoded. XML payloads of document/literal style messages can be inserted directly into the message body with a default namespace.

4.3.1.2 Payloads as Attachments

Network Nodes must support Direct Internet Message Encapsulation (DIME) [See reference 14]. DIME is a binary Protocol with better performance compared to the SOAP with Attachment Protocol [See reference 12].

It is expected that more SOAP stacks will provide support for both Protocols, in such a way that the attachments are decoded at the transport level.

4.3.2 Payload Validation

The Network Exchange Protocol V1.0 does not govern payload issues. However, it is expected that all XML payloads will be validated using the XML schema-based DETs located in the Network registry that are used to XML encode the payload.

4.3.3 Payload Compression

XML payload can be compressed using a number of different techniques. Most compression techniques, when applied to XML, typically achieve very high compression ratios. However, XML compression changes the structure of an XML document, which complicates the process of digital signature (An XML document requiring signature must have a canonical form in order to be processed correctly by both the signer and the verifier). Therefore, compression of SOAP messages will not be permitted at this time. However, Network Nodes are encouraged to support attachment compression to improve Network performance.

5.0 Network Services

A Protocol defines the structure of an interaction that will take place among two or more parties. It defines the rules that must be followed by each of the parties in order for them to successfully fulfill their role in the interaction.

The Network Exchange Protocol V1.0 will involve a series of interactions or conversations among the various Network trading partners and business components. These conversations will generally consist of service requesters (i.e. other Nodes or simple clients) requesting services of service providers (Nodes). The service requests will primarily involve requests for information from a Web service, which will then typically respond with the requested information or a fault message of some type. All service requests will utilize the message structure defined above. All requests and responses will be encoded using SOAP.

However, the conversations between Network parties can be much more complex than simple request/response, with different parties initiating the conversation or taking up requests and responses at different points in the process to accomplish different objectives.

5.1 Conversation Structure

The conversations moving across the Network will be composed as depicted in Figure 6. All messages will be built on a basic set of operational primitives. These primitives will be used to construct the basic exchange service interactions. These service interactions will then be strung together to implement entire business processes associated with the exchange of environmental data. For example, the process of one state collecting weekly water monitoring results from a neighboring states Node is an Exchange Business Process as would be EPA collection of monthly activity reports for a delegated program.

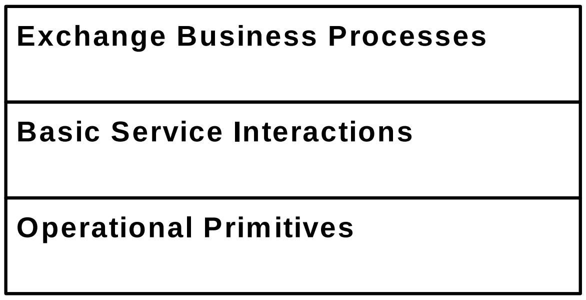


Figure 6 - Network Exchange Conversation Structure

Note that the Protocol and Specification focus on the two lower layers of this conversation. Definition and documentation of larger Exchange Business Processes is being recommended as an immediate follow on activity to the Network Node Functional Specification V1.0 – this effort will be informed by the early implementation experience implementing flows with the Network Exchange 1.0 Protocol.

5.2 Basic Message Configurations

As discussed in the previous section, SOAP messages are the basic currency of the Protocol. There are four fundamental ways that messages can be arranged, each is called a Message Configuration. All Network service interactions (i.e. the submission of data or a query) are constructed from one of these basic message configurations. The Network service interactions are described in the next section.

5.2.1 Request/Response

In a request response configuration, a message is sent from the service requester to the service provider, and either a response or a fault is received from the service provider.

5.2.2 One-Way

In a one-way configuration, a message is sent from the service requester to the service provider, and no response or fault is allowed from the service provider.

5.2.3 Solicit Response

In a solicit response configuration, a message is sent from the service provider to the service requester, and either a response or a fault is received from the service requester.

5.2.4 Notification

In a notification configuration, a message is sent from the service provider to the service requester, and no response or fault is allowed from the service requester.

5.3 Response Types

There are five (5) different responses that can be received from a service provider in response to a request.

1. Simple Response
2. Receipt Acknowledgement
3. Notification
4. Solicit Response
5. Error

5.3.1 Simple Response

The simple response will have the return value encoded in the body of the response SOAP message. The return value will be a single structure. The convention is to name the message response structure with the name of the request with the string, “Response”, appended to it.

5.3.2 Receipt Acknowledgement

Some requests will provide an immediate acknowledgement response to the service requester of request receipt by the service provider. This receipt acknowledgement will not contain the data being returned as a result of the request. Instead, this information will be returned in a separate subsequent “notification” message sent from the service provider to the service requester. This type of interaction is necessary in situations where the generation of the regular return value may take considerable time, and the service requester needs to verify whether the request was successfully received by the service provider. Use of receipt acknowledgement is determined by the specific dataflow and the governing procedures and policies of the flow. A Node may be required to send an email acknowledgement when a request is processed.

5.3.3 Notification

A notification is sent from the service provider to the service requester. This will typically be as a follow-on message to an initial service request that had a receipt acknowledgement sent from the service provider to the service requester. These notifications will typically contain large payloads that are being returned by the service provider to the service requester in response to the initial service request. It is possible that some very large payloads may be broken up into smaller payload for transmission. In this case, multiple notifications may be necessary.

5.3.4 Solicit Response

Instead of a notification, a solicit response may be returned after an acknowledgement. This might be used in cases where the service requester may have requested a scheduled or otherwise conditional response rather than an immediate response. The service provider may send a solicitation back to the service requester asking whether the conditions are acceptable to the service requester, and expect to get a response from the service requester in reply. If the condition were acceptable to the service requester, then the service provider would subsequently send a notification containing the return data.

Another type of solicit response might occur in a situation where a requested service was unavailable at the time of the request for some reason. The service provider may reply that the request will be processed at some point in the future when resources are available, and would solicit the service requester's acceptance of this.

5.3.5 Error

Any error condition would be returned as a SOAP fault. There are many types of errors that could occur. Errors are further detailed in Section 0.

5.4 Basic Network Service Interactions

The Network is Web services architecture. As the name implies, the Network is made up of basic services that interact to fulfill business exchanges. The Protocol uses the term "Basic Network Service Interactions" to describe how the sets of messages, configured in one of the four ways described above, get something done. These service interactions are the heart of the Network Protocol and the operation of the Network itself. These service interactions are described below: Note that this section does not cover message structures and functional details of the service interactions (See Network Node Functional Specification.)

The following are the Network exchange service operations:

- Authenticate
- Submit
- GetStatus
- Query
- Solicit
- Execute
- Notify
- Download
- NodePing
- GetServices

5.4.1 Authenticate

Authenticate is the first method a client calls in order to gain access to the Network exchange service. Users must supply an identification (userId) and a credential; the service provider returns, upon a successful authentication, a ticket, known as the securityToken. The securityToken is required for all subsequent Network service interactions. The topic of using securityToken for access control is further discussed in the Security section. Authenticate is a request/response message configuration.

5.4.2 Submit

The Submit method allows a client to send documents (of various formats) to the Network service (typically a partner Node). The document in the request message is formally defined, using XML schema, as:

```
<complexType name="nodeDocument">
      <sequence>
        <element name="name" type="xsd:string"/>
        <element name="type" type="xsd:string"/>
        <element name="content" type="xsd:string"/>
      </sequence>
</complexType>
```

Where the sequence tag indicates that all child elements must be in sequential order as specified. As can be seen in the schema segment, each document has a name, a type (XML file, Flat text, etc), and contents. Document contents are either embedded in the message body as base64-encoded string (of type base64Binary), or a reference to an attachment associated with the request message.

The request message, as noted previously, must contain a securityToken issued by the Node or an authentication server. It must also include a predefined dataflow identifier. The request message is defined in the Node 1.0 WSDL segment as follows:

```
 <message name='Submit'>
    <part name='securityToken' type='xsd:string'/>
    <part name='transactionId' type='xsd:string'/>
    <part name='dataflow' type='xsd:string'/>
    <part name='documents' type='typens:ArrayofDoc'/>
  </message>
```

The documents element in the request message is an array of nodeDocuments.

Once a preliminary process is completed successfully, the service provider returns a transaction ID, which can be used to track the status of the submission.

The whole transaction (i.e., the submission service interaction) fails if any one of the documents could not be processed successfully. The service provider should return a SOAP fault detail indicating the name of the failed document, but the message should be interpreted as the failure of the whole submission.

Submit is a request/response message configuration.

5.4.3 GetStatus

The method is used to query the status of a previous transaction. The requester sends the message along with a transaction ID obtained from a Network Node.

The initial Protocol 1.0 list of status responses is:

- Received: A submission was received by the service but has not been processed.
- Pending: One or more documents are to be downloaded by the service.
- Processed: The submission has been processed by the Node, but waiting to be delivered to its ultimate destination (i.e. a partner system or another Node).
- Completed: The submission is complete and accepted by the target Node.
- Failed: The submission has failed. The requester should resubmit.

This list will be expanded as needed.

A dataflow may have program-specific statuses understandable by submitters. The following diagram shows a general state transition of status for a typical document submission:

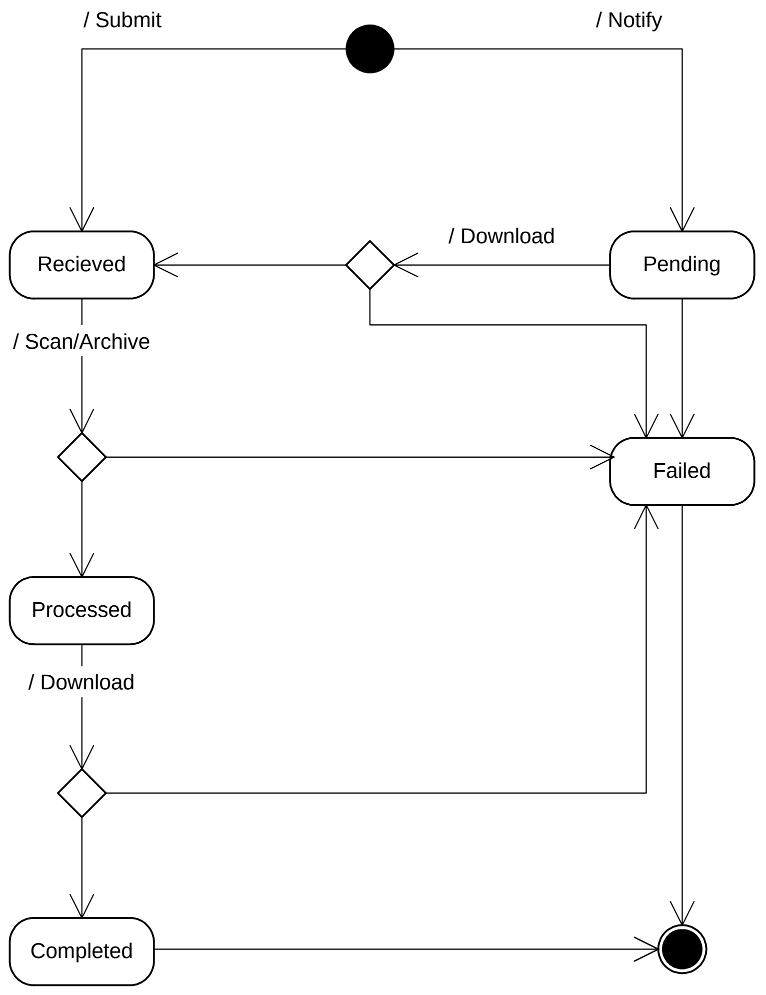


Figure 7 - State Transition Diagram for Document Submissions

5.4.4 Query

Query is a request/response message configuration. The method provides a capability to query data on a partner Node and receive back XML encoded data. It has the following parameters:

```
<message name='Query'>
    <part name='securityToken' type='xsd:string'/>
    <part name='request' type='xsd:string' />
    <part name='rowId' type='xsd:integer/>
    <part name='maxRows' type='xsd:integer/>
    <part name='parameters'type='typens:ArrayofString'
  </message>
```

where

- securityToken (required): An authentication token previously returned by the authenticate method.
- request (required): The database logic to be processed it contains the name of a service request or a stored procedure.
- parameters (optional): An array of parameter values.
- rowId: The starting row for the result set, it is a zero based index to the current result set.
- maxRow: The maximum number of rows to be returned.

The service provider returns a result set, bound by a schema associated with data, when successful.

5.4.5 Solicit

The Solicit method is designed for facilitating asynchronous Query operations. When a Query request takes long time to execute, the method allows a requester to trigger the operation and to download the result later when done.

Asynchronous operation using the Solicit method is further discussed in Section 0.

5.4.6 Execute

Execute is a request/response message configuration, optional in the V1.0 implementation. The method provides the capability to request a partner Node to execute some operation (e.g. a database operation or other business process) based on the request and parameter values. It has the following parameters:

```
<message name='Execute'>
    <part name='securityToken' type='xsd:string'/>
    <part name='request' type='xsd:string' />
    <part name='parameters'type='typens:ArrayofString'
  </message>
```

where

- securityToken: An authentication token previously returned by the authenticate method.

- request: The name of an operation (e.g., process or stored procedure to be executed).
- parameters: An array of parameter values for the request.

The service provider returns a response message that contains the number of rows affected by the operation.

Execute, together with the Query method, can be used to create a database-programming environment where the target databases are distributed across the entire data exchange Network. Integration of heterogeneous database information becomes possible because the interface encapsulates the specifics of native database systems.

However, because (among other uses) Execute can be used to pass raw Structured Query Language (SQL) or other statements, requires much higher access privileges than any other operations, Network Nodes must make sure that the person requesting the service is not only authenticated, but also authorized to perform the operation.

Execute is an optional operation. Service providers are should implement Execute using raw SQL only if the following features are also supported:

- Universal Data Access: Capable of handling database operations regardless of the DBMS. A standard approach, such as ODBC or JDBC, should be used to implement the interface.
- Multi-level Authorizations: User access rights (operation privileges) are verified at Network service level based on a predefined security policy. The rights (object privileges, i.e., rights to access a table) are further validated by the Database Management System (DBMS).

5.4.7 Notify

This method has three intended uses:

1. Document Notification: Notify of changes, or availability, of an array of documents to a Network Node.
2. Event Notification: Send Network events to peer Nodes. The semantics of Network events are application specific.
3. Status Report: Notify the processing status of a previous service interaction to a requester.

5.4.7.1 Document Notification

The ***notify*** method is different from ***submit*** in that there are no document contents, or attachments in the request message. The message simply informs a Network Node that some documents are ready to be retrieved; the service provider can, at its own convenience, download them at anytime.

The format of the message is defined by the following WSDL segment:

```
<message name='Notify'>
  <part name='securityToken' type='xsd:string'/>
  <part name='dataflow' type='xsd:string'/>
  <part name='documents' type='typens:ArrayofDoc'/>
</message>
```

Each document in the message contains a URI pointed to the resource.

In addition to a transaction ID, which is returned immediately, the service provider is required to send an acknowledgement to the requester through email or a client provided callback method when the documents are downloaded, be it successful or not.

It is highly recommended that service providers use a quality control strategy to detect transmission errors early, and retry multiple times when necessary. Nodes are required to provide detailed transaction logs that contain all transaction records, either succeeded or failed. It is also recommended that activity logs be provided so that problem tracking and debugging are possible.

Partners may also use Notify to alert internal Nodes (i.e. destination systems) that a document has been successfully received, scanned and archived and is ready for loading. EPA's CDX is considering this approach to alert its program system customers that documents are ready for loading.

5.4.7.2 Event Notification

The Notify method can also be used for sending event notifications. The Protocol uses:

```
http://www.exchangenetwork.net/node/event
```

as a unique identifier for Network events. If the dataflow element in a request message contains the above URI, then an event has occurred in the Network and the nodeDocument structure contains the type and description of the event.

The event URI can be extended to carry additional information if needed. For instance, a Node failure event can be represented by:

```
http://www.exchangenetwork.net/node/event/down?node=5
```

which indicates that Node 5 in the Network is not functioning properly.

A standard list of Network events will be established based on early implementation experience of this Protocol.

5.4.7.3 Status Notification

Status notification is a Solicit Response operation. A service provider may notify a requester of process status, i.e., file submission status, using the same notification message. A notification is a status notification if the dataflow element in the request contains a prefix of:

```
http://www.exchangenetwork.net/node/status
```

The name of the document should be the transaction ID, and the content element contains a status string.

Status notification is a complement of the GetStatus operation in that submission (or operation) status information can flow both ways. In some situations when documents have to go through a lengthy process, an impatient submitter may call GetStatus many times with no expected result. With status notification, however, the submitter is notified when the status of the submission changes. Active status notification can, in many situations, reduce Network traffic and improve the quality of services.

5.4.8 Download

This method is a callback function for data exchange. After being notified of the availability of a set of documents (through either the **Notify** method or other means) or per a pre-established schedule, the service provider needs to download and process the updated files. The download method initiates a request from the service provider for the documents,(i.e., document pulling). The role of the Network Node switches from a service provider to a requester at the moment of the request.

Note that pulling can, depending on the nature of dataflow, be on demand or scheduled. Download operation can take place without prior notification in some exchange scenarios where document location and availability are predefined.

The Download method is a complement of Submit in that it facilitates bi-directional dataflows between Nodes. In other words, a Network Node can be a sender at one time, but a receiver at another. With Download and Submit, the Node Network becomes symmetrical from a dataflow point of view. The following dataflow diagram shows a symmetrical Network with three participating Nodes. The Download dataflows inbound from the requester point of view; the Submit dataflows outbound.

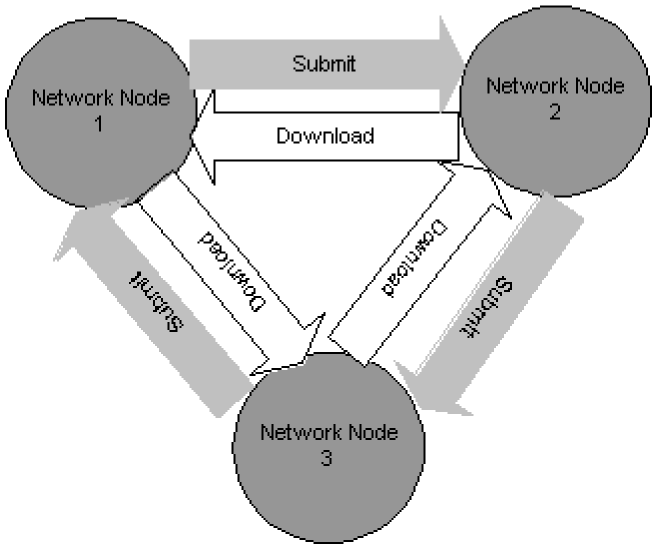


Figure 8 - Bi-directional Flow Diagram with Submit and Download

Download is a solicit response message configuration.

5.4.9 NodePing

The NodePing method is designed for checking the availability of a Network Node. A Network Node is not available if:

- A connection to the Node cannot be established. The NodePing method on the client side would generally produce a Network exception. There will be no response from the method call.
- The response message is a SOAP fault, with a status code 500 for HTTP transport. This indicates that, although the server is up and running, it is not ready for Network exchange services at this time.

5.4.10 GetServices

The GetServices method is an administrative function for examining the capability of a Node. Due to the dispersed nature of the Network, a Node may elect to support only part of the functionality defined in the Protocol. The GetServices method allows a requester to query the following features:

- Interfaces: A list of interfaces supported by the Node, see the Node Functional Specification for details on interface definitions.
- Query: Database queries supported by the Node.
- Execute: Data manipulation capability provided by the Node.

5.5 Network Exchange Business Processes

Partners will establish Network Exchange Business Processes by combining Network service interactions (e.g. authenticate and then download) The following scenarios outlines typical ways services can be combined. They document who requests what of whom, and what kind of responses can be expected.

Example Scenario	Example Usage
Simple Document Submission	State Node transmits monthly report to EPA CDX.
Requested Download	State Node notifies EPA/CDX of the availability of a monthly report for download.
Sending Network Events	Node notifies a trading partner that it is going down.
Broadcasting Network Events	Node notifies multiple trading partners that it is going down.
Retrieving Information with Query	Client application queries a Node for “drill down” information on one monitoring location.
Executing SQL Procedure	Two trading partners interchange database records.
Performing Asynchronous Operations	One partner routinely requests a large or complex query from a partner Node, which the partner services as resources permit.

Note in the scenario examples described below, the process of token validation is omitted for brevity. Initially all flows with EPA will use the Network Authentication and Authorization Service (NAAS) for token validation, Network partners can use the NAAS for other flows and/or may establish their own local security servers.

Simple Document Submission

In a simple document submission operation, a client wants to send an array of documents (i.e., one or more) for a specific dataflow to a Network Node. The procedure is outlined below:

1. The client sends an Authenticate message, with user ID and credential, to the Node; the service provider returns a securityToken after successful authentication.
2. The client invokes the Submit method with a set of documents. If successful, the service provider returns a transaction ID for status tracking.

3. Optional. The client queries submission status using GetStatus, and resubmits if failed.

The whole process is represented in the following Unified Modeling Language (UML) activity diagram:

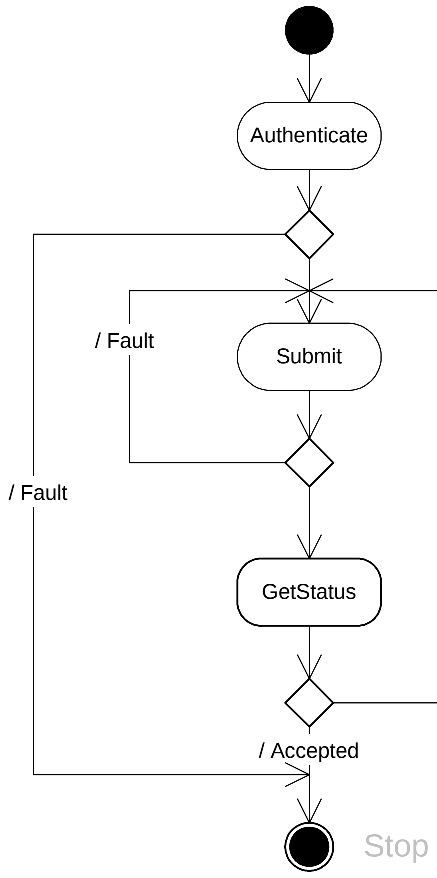


Figure 9 - UML Activity Diagram for Simple Submissions

The diagram indicates that the client can resubmit the document if the submission failed during flow specific processing.

Note that if the client invokes the GetStatus method at a time when the securityToken has expired, it must call the Authenticate method again to obtain a valid securityToken.

Figure 10 shows a UML sequence diagram for simple document submissions. The requester and the service provider are in synchronized operation mode using the request/response exchange model.

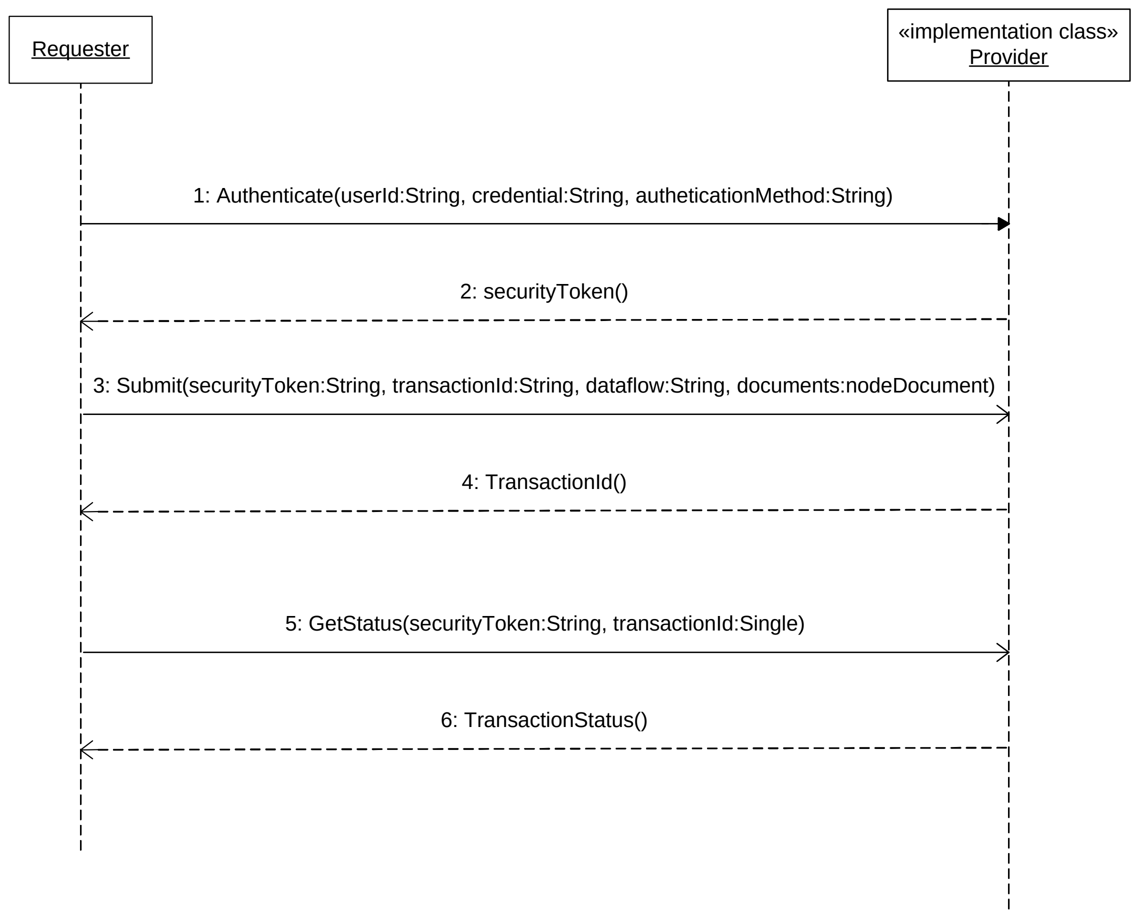

Requester
«implementation class» Provider
1: Authenticate(userId:String, credential:String, autheticationMethod:String)
2: securityToken()
3: Submit(securityToken:String, transactionId:String, dataflow:String, documents:nodeDocument)
4: TransactionId()
5: GetStatus(securityToken:String, transactionId:Single)
6: TransactionStatus()

Figure 10 - UML Sequence Diagram for Document Submissions

5.5.1 Requested Document Download

In solicited operations, a client notifies a Node of the availability of some document. The service provider returns sometime later and downloads the specified document as requested.

Solicited operations help the service provider to avoid peak conditions. Documents can be transferred at a preferred time when traffic is relatively light.

Solicited operations are not limited to client and server. They can also occur between service providers, and more importantly between an intermediate Node and the ultimate document receiver.

A typical solicited operation is presented in Figure 11. One interesting phenomena in the operation is that, after a successful notification, the requester and the provider run in parallel. The provider may be in the process of downloading the documents while the requester is checking the status of the transaction. The shaded boxes in the diagram represent processes on the service provider side.

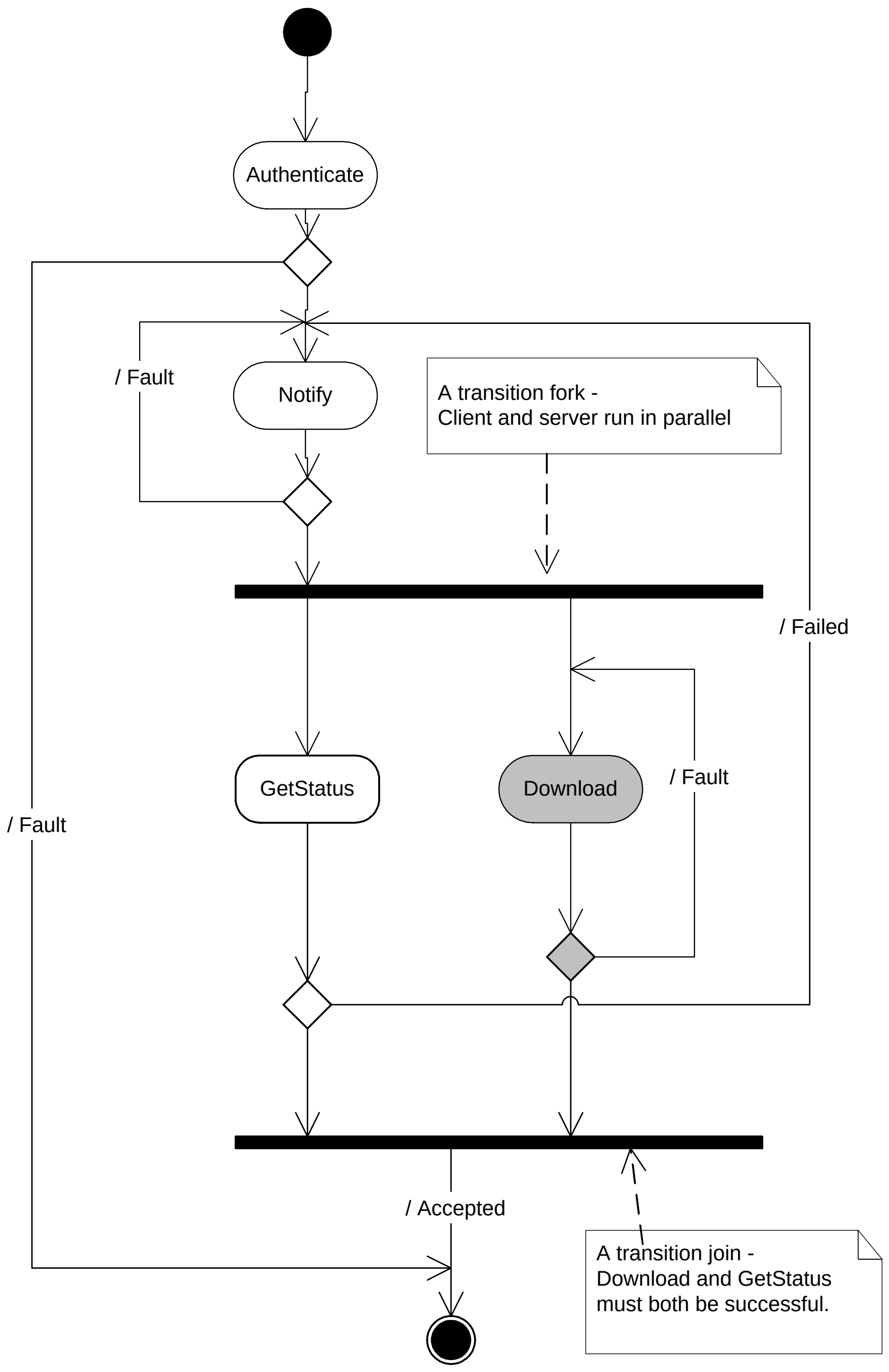


Figure 11 - UML Activity Diagram for Solicited Operations.

The sequence of download operation is further illustrated in Figure 12. The process is outlined below:

1. Node A sends an authenticate message to the Node B.
2. Node B returns a securityToken if authentication is successful.

3. Node A invokes the Notify method and informs Node B about availability of a set of documents.
4. Node B acknowledges the notification and returns a transaction ID for status tracking.
5. Some time later, perhaps when Node B has idle time, it initiates a download operation by authenticating itself with Node A.
6. Node A returns a securityToken, granting access to Node B.
7. Node B sends a download message to Node A, asking for the documents.
8. Node A embeds or attaches the documents in the response message, and sends it.
9. To verify transaction status, Node A may call the GetStatus method to check the status of the submission.
10. Node B delivers the status string in the response message.

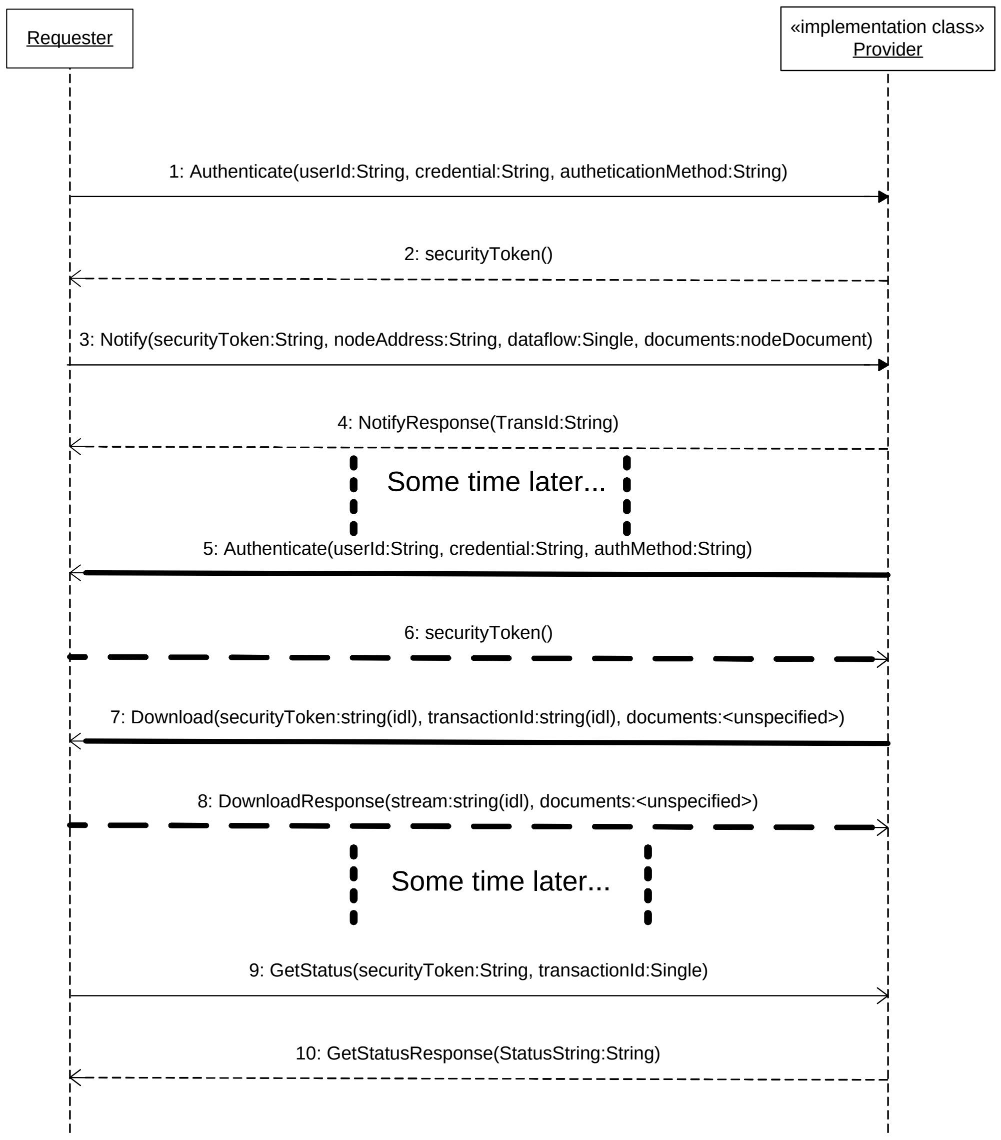

Requester
«implementation class» Provider
1: Authenticate(userId:String, credential:String, autheticationMethod:String)
2: securityToken()
3: Notify(securityToken:String, nodeAddress:String, dataflow:Single, documents:nodeDocument)
4: NotifyResponse(TransId:String)
Some time later...
5: Authenticate(userId:String, credential:String, authMethod:String)
6: securityToken()
7: Download(securityToken:string(idl), transactionId:string(idl), documents:<unspecified>)
8: DownloadResponse(stream:string(idl), documents:<unspecified>)
Some time later...
9: GetStatus(securityToken:String, transactionId:Single)
10: GetStatusResponse(StatusString:String)

Figure 12 - UML Sequence Diagram for Download Operations

5.5.2 Sending Network Events

Sending a Network event is different from other operations in that the sender does not care about receiving a response, i.e., it is typically a one-way operation. If the underlying transport is HTTP/HTTPS, however, the receiver must send a response for it to be successful. This is because HTTP is a request-response Protocol in which lack of a response is treated as a Network error. Nevertheless, the receiver can safely discard the response message, as it carries no semantic meaning.

A Network event is modeled using the Notify message (See the Functional Specification for details). If the **dataflow** argument in the message is a URI:

```
http://www.exchangenetwork.net/node/event
```

then the message is an event. The document structure inside the message shows the type and description of the event.

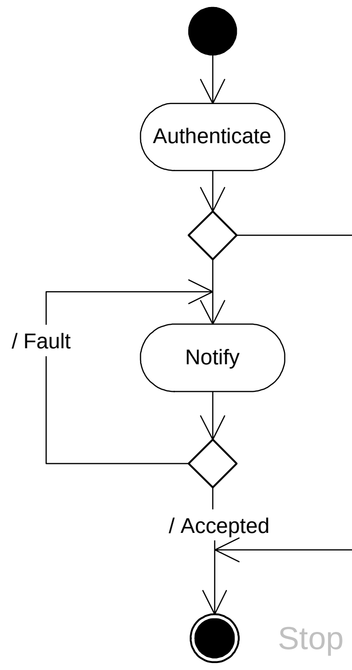


Figure 13 - UML Activity Diagram for Event Notifications.

5.5.3 Broadcasting Network Events

A broadcast operation is an operation that sends an event to one or more Nodes, either sequentially or concurrently. For a broadcaster to send such an event, it must know who is interested in the event and where to send the message (listeners.) The Network Exchange Protocol does not specify how this should be accomplished.

The following section describes how Broadcast is envisioned to work once a Network UDDI registry is established. This discussion is not a normative part of the Protocol. In the Network Node configuration, an event is registered as a tModel (a technical fingerprint) in the UDDI registry. Nodes who are willing to be notified will then create a service that supports the tModel, which is the equivalent of saying:

- Let me know when the event happens, and call me at this endpoint.

So when the broadcaster searches for Web services that support the tModel in the UDDI registry, it gets a complete list of all listeners. Since the broadcaster knows the exact format of the Notify message, it is a simple matter to send the same message to everyone in the list. The whole process is shown in Figure 14.

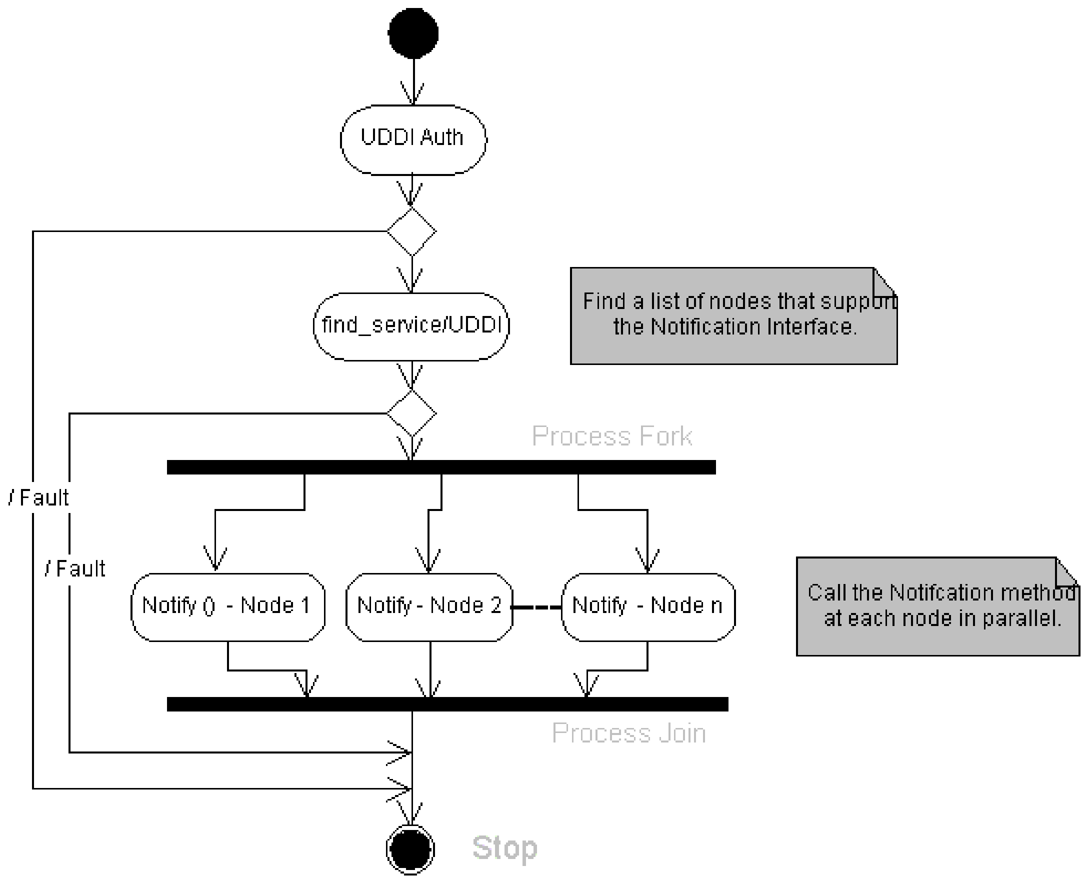


Figure 14 - UML Activity Diagram for Event Broadcasting.

5.5.4 Retrieving Information using Query

This Protocol defines a simple method, Query, for all Node queries. In a typical operation, a service provider would create named reports, or predefined information requests on the database server. The client sends a Query request message, including associated parameters, indicating which report or procedure to execute. A response with all selected records is returned.

Given the generic database query capability, it is entirely possible to move relational data from one Node to another. For instance, Node A may query daily updated records on Node B and insert, after mapping to its own data elements, the updated records into another table. The operations can all be conducted automatically, either by schedule or by a triggering event.

Early implementations of query will likely consist of honoring a small set of standardized requests associated with a given Network flow. However given the inherent flexibility of the approach, and the creativity of Node implementers, Nodes could easily mount an ever-expanding range of queries and respective sources.

Figure 15 is a simple activity diagram for the Query operation. The diagram assumes that the requester knows what statements or procedures the provider supports. Given the discussion above, this may not be true in all situations due to the dynamic nature of Web services. A Node may suspend support for certain queries at one time, or add more queries at another. The Protocol defines a method, GetServices, for querying currently available data requests at a

Node. When invoked with Query as a parameter, the method returns a complete list of all requests available at that time.

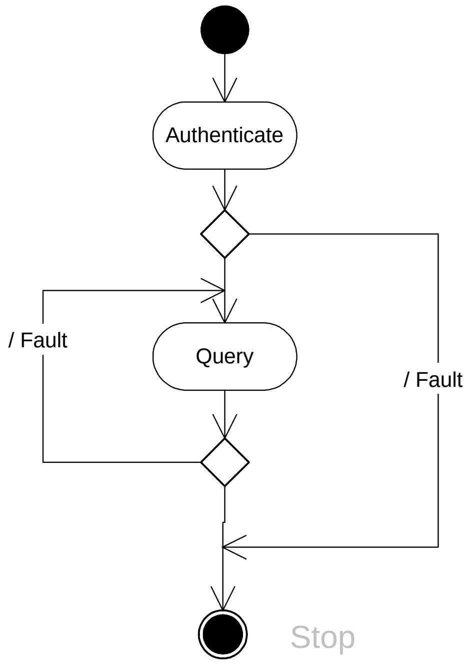


Figure 15 - UML Activity Diagram for Simple SQL Queries

Figure 16 shows a sequence diagram for the Query operation. The requester, in this case, asks the provider for a list of available queries. The requester Node then sends a Query message using one of the queries from the list, and gets a result set back.

Although the GetServices method can be used to discover what a Node has to offer at run-time, the results are not in a form that is semantically meaningful to machines. This is where the FCD may come into play. The parties can establish not only rules for conducting queries, but also the forms of the query, the required parameters, and the expected results. The Node 1.0 team deliberated options for machine readable responses from GetServices but decided that the pace of external standards development (especially WSDL/1.2) did not justify the added complexity. As queries proliferate a means of jointly managing them will need to be developed/adopted.

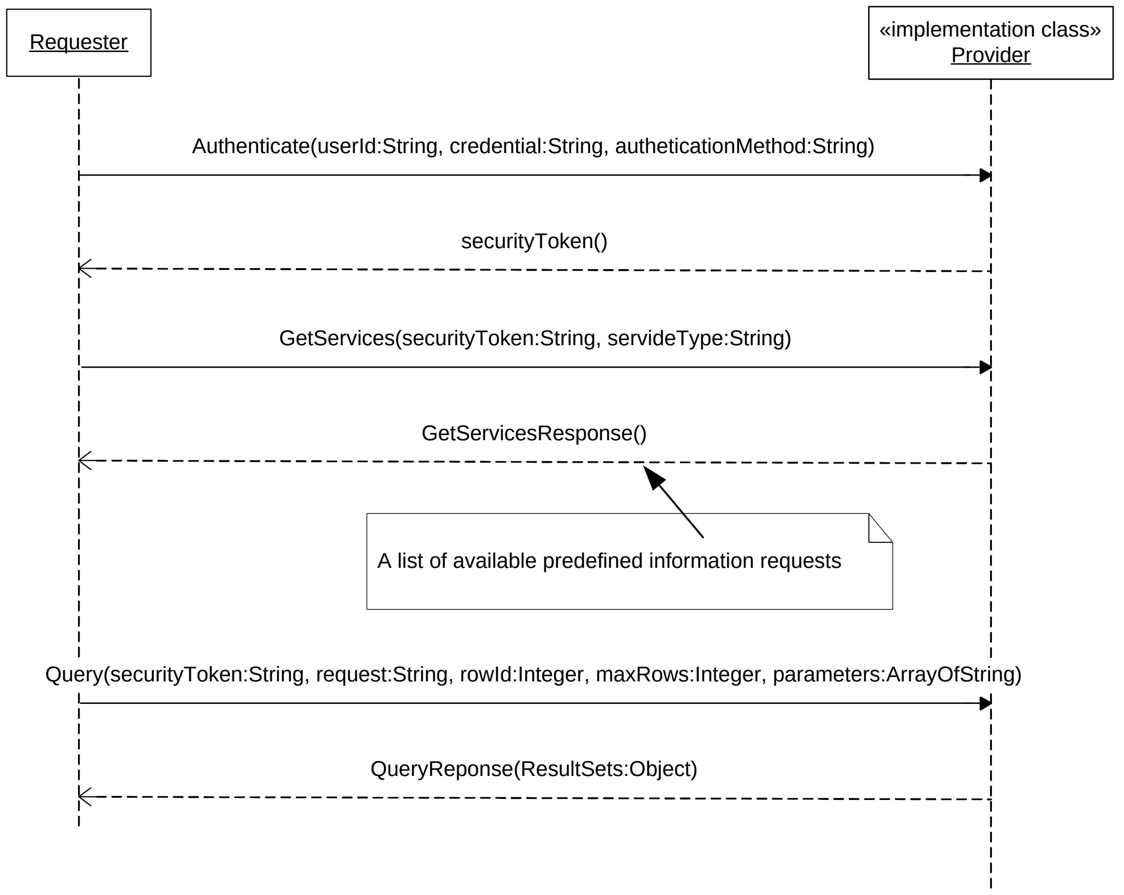

Requester
«implementation class» Provider
Authenticate(userId:String, credential:String, autheticationMethod:String)
securityToken()
GetServices(securityToken:String, servideType:String)
GetServicesResponse()
A list of available predefined information requests
Query(securityToken:String, request:String, rowId:Integer, maxRows:Integer, parameters:ArrayOfString)
QueryReponse(ResultSets:Object)

Figure 16 - UML Sequence Diagram for Query Operations

5.5.5 Executing predefined Procedures

The Execute method is designed for data manipulation using predefined procedures, or other Node specific services that could not be handled by the Query method.

The procedure for executing a predefined procedure or a service request is outlined below:

1. The client sends an Authenticate message to logon to the Network.
2. The client invokes the Execute method, passing all data to the service provider.
3. The service provider processes the requested procedure and returns a status of the execution.

The procedure is shown in the following sequence diagram:

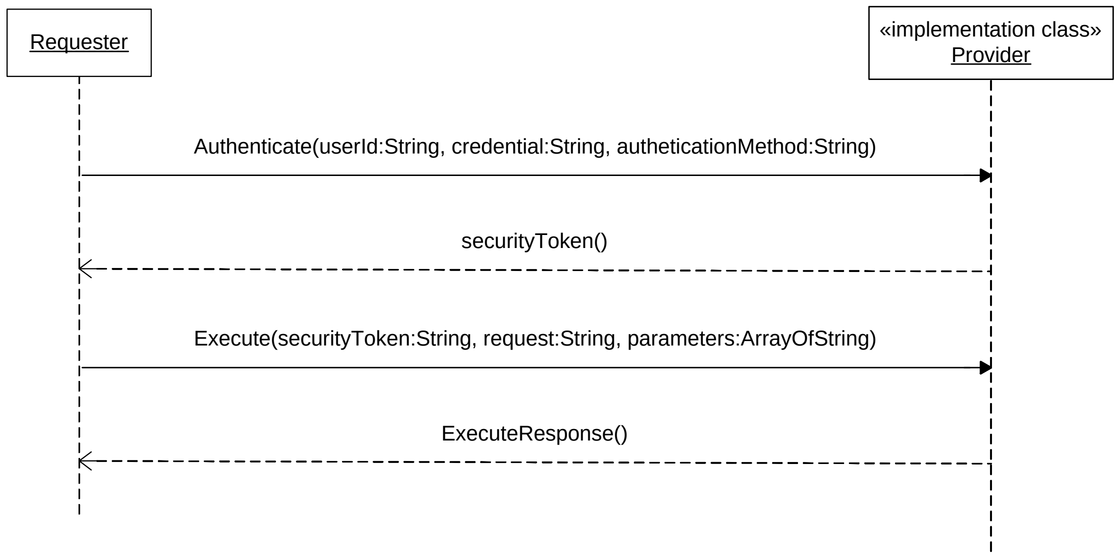

Requester
«implementation class» Provider
Authenticate(userId:String, credential:String, autheticationMethod:String)
securityToken()
Execute(securityToken:String, request:String, parameters:ArrayOfString)
ExecuteResponse()

Figure 17 - UML Sequence Diagram for the Execute Operation

5.5.6 Performing Asynchronous Operations

This section discusses some of the basic configurations and scenarios for asynchronous operation using the Solicit method.

5.5.6.1 Network Configuration

Asynchronous data exchanges can take place in different ways based on the Network configuration:

1. Pure Client: In this scenario, a requester (a client application) wants to conduct an asynchronous operation with a Network Node. Because it is a pure client with no Node full implementation, it is the client responsibility to check the status of the transaction and download the document when available. The sequence of operations in this case is Solicit-GetStatus-Download.
2. Network Node: This is the case where one Node, say Node A, (or a requester at the Node) asks another Node, Node B, to perform an asynchronous operation. After the operation is completed, Node B submits the result set to Node A. The sequence of operations in this case is Solicit-Submit. Since Node B is in the best position to know when the operation is done, it can send the result to the target Node as soon as possible.

5.5.6.2 Procedures of Asynchronous Exchanges

5.5.7 Pure Client Interactions

This scenario is analogous to ordering a pizza, for pickup from a busy parlor. The patron views the menu (the list of supported solicit requests) orders the pizza by phone (e.g. "one number 17 pepperoni parameter"), the parlor responds that you are order number 1234, and that you should call back to check the status of the order. The parlor makes the pizza and stores it on the rack, the patron calls back, gives the order number and is told the pizza is ready for pickup (download).

Figure 18, UML sequence diagram shows a typical exchange under such situations:

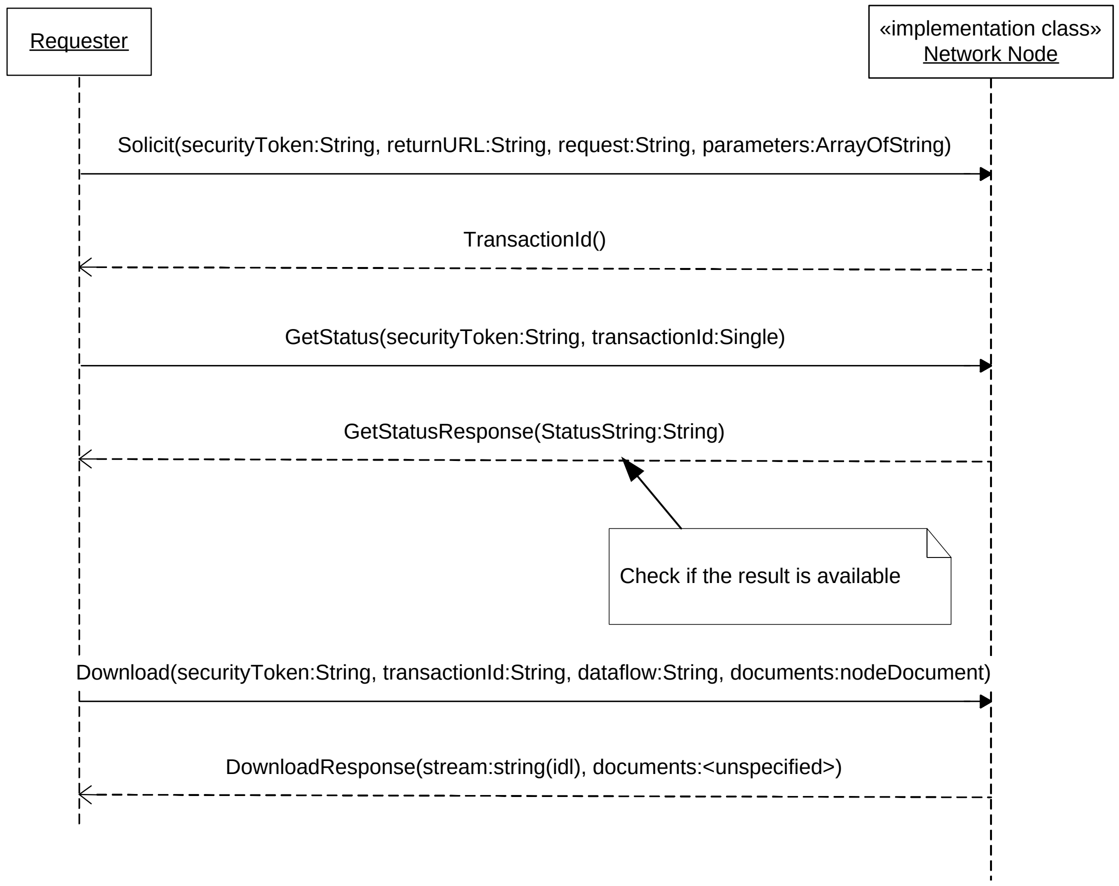

Requester
«implementation class»
Network Node
Solicit(securityToken:String, returnURL:String, request:String, parameters:ArrayOfString)
TransactionId()
GetStatus(securityToken:String, transactionId:Single)
GetStatusResponse(StatusString:String)
Check if the result is available
Download(securityToken:String, transactionId:String, dataflow:String, documents:nodeDocument)
DownloadResponse(stream:string(idl), documents:<unspecified>)

Figure 18 - UML Sequence Diagram

The procedure is outlined as follows:

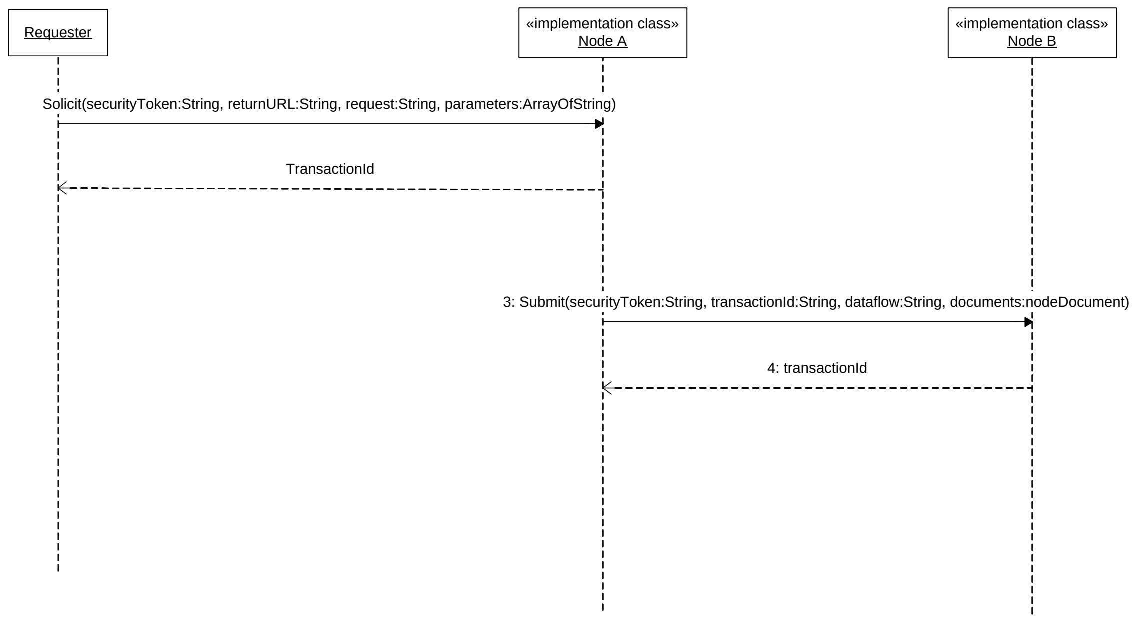
1. The requester sends a Solicit message to the provider, specifying the stored procedure to be executed and its parameters. The return URL parameter is set to empty because there is no Node implementation at the requester side.
2. The provider marks the transaction as pending and returns a transaction ID immediately.
3. The provider processes the transaction some time later, and set the status of the transaction to either Completed or Failed based on the final result.
4. Meanwhile, the requester may occasionally check the status of the transaction by invoking the GetStatus method.
5. The requester downloads the document when the transaction is completed successfully. It may retry the whole procedure if failed.

5.5.7.1 Network Node Interactions

Using our pizza example, this is perhaps the most common scenario:

> I would like a medium supreme pizza to be delivered to the CDX node.

In this configuration, Node A is not only a service provider, but also a requester, which allows it to deliver results to the target address. The UML sequence diagram is shown in Figure 19.



Requester
«implementation class» Node A
«implementation class» Node B
Solicit(securityToken:String, returnURL:String, request:String, parameters:ArrayOfString)
TransactionId
3: Submit(securityToken:String, transactionId:String, dataflow:String, documents:nodeDocument)
4: transactionId

Figure 19 - UML Sequence Diagram for Requester and Provider

The procedure is outlined as follows:

1. The requester sends a Solicit message to Node A, specifying the stored procedure to be executed, its parameters and the return URL - Node B (the delivery address).
2. Node A marks the transaction as pending and returns a transaction ID immediately.
3. Node A processes the query some time later.
4. If successful, Node A submits the result to Node B as requested. It sets the status of the transaction to either Completed or Failed based on the status of the final submission.

5.5.7.2 Using Network Authentication and Authorization Services (NAAS)

NAAS are centralized security services. Security tokens and assertions issued by NAAS are trusted and accepted by all Network Nodes. In order to jump-start the Network, EPA has agreed to host the initial version of the NAAS. This will allow Network partners the opportunity to implement the Protocol as next generation security technologies and services are established and validated.

NAAS provide a set of standard Web services across the Network, which can be easily accessed by Network users and services providers. All operations defined in NAAS must be conducted over a secure SSL channel using 128-bit encryption.

5.5.8 Network Authentication

5.5.8.1 Direct Authentication

In a direct authentication, the requester sends an Authenticate message to the NAAS and obtains a securityToken. Steps of direct authentication are outlined as follows:

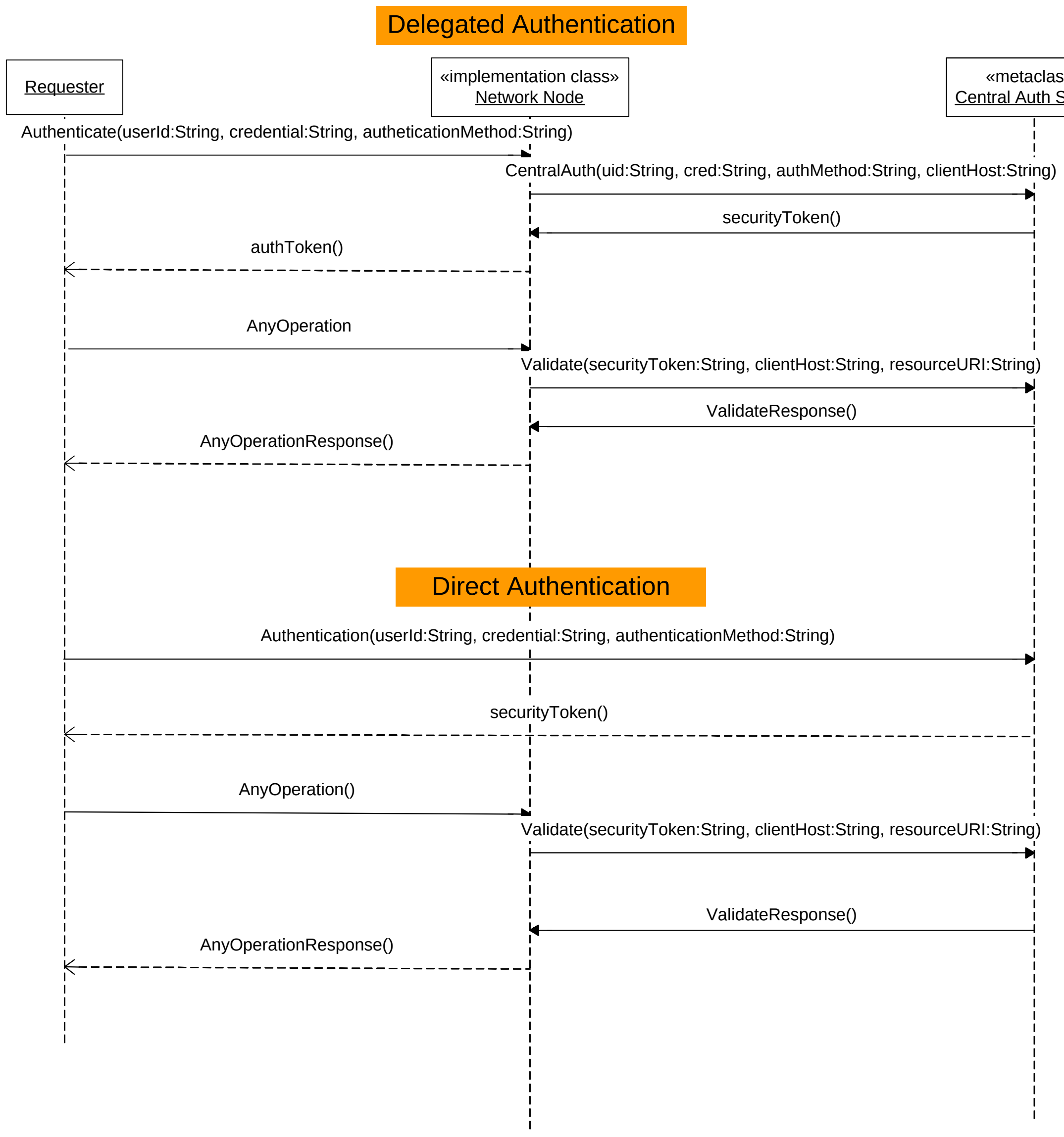
1. The client sends an Authenticate message to NAAS, and obtains a securityToken when successful.
2. The client then sends a request to a Network Node (Node A, for instance) along with the securityToken.
3. Node A sends the securityToken to NAAS for validation and authorization.
4. The NAAS service verifies the securityToken. It returns a SOAP Fault message when validation fails and a positive response when validation succeeds.
5. Node A performs the operation only when the NAAS response is positive.

5.5.8.2 Delegated Authentication

In this application scenario, the requester sends an authentication message to a Network Node. The Node then delegates the authentication request to the NAAS for processing.

This model simplifies client interactions with a Network Node because the client can perform all tasks at the single entry point (with a single WSDL file, perhaps). However, a small performance impact is expected because the overhead of routing the message to NAAS.

The following UML sequence diagram (Figure 20) shows interactions between the requester, the Network Node and the NAAS.



Delegated Authentication
Requester
«implementation class» Network Node
«metaclass» Central Auth Service
Authenticate(userId:String, credential:String, autheticationMethod:String)
CentralAuth(uid:String, cred:String, authMethod:String, clientHost:String)
securityToken()
authToken()
AnyOperation
Validate(securityToken:String, clientHost:String, resourceURI:String)
ValidateResponse()
AnyOperationResponse()
Direct Authentication
Authentication(userId:String, credential:String, authenticationMethod:String)
securityToken()
AnyOperation()
Validate(securityToken:String, clientHost:String, resourceURI:String)
ValidateResponse()
AnyOperationResponse()

Figure 20 - Using the Network Authentication and Authorization Service

5.5.9 Network Authorization

Authorization is a process of granting access to resources to a user based on a certain access control policy. Given the authenticated user identity (the subject) and the security policy of a Network resource (the object), the central authorization server would determine whether or not to grant access. The authorization service answers the following question:

Is operation X by principal Y on resource Z permitted?

NAAS performs the entitlement checking operations using a Web method – Validate. The request message of the method is defined as follows:

```
<message name='Validate'>
    <part name='securityToken' type='xsd:string'/>
    <part name='client Host' type='xsd:string'/>
    <part name='resource URI' type='xsd:string'/>
  </message>
  <message name='Validate Response'>
    <part name='return' type='xsd:string'/>
  </message>
```

The service returns an OK message when the subject is authorized, a SOAP fault message otherwise.

Additional details of use of the NAAS for authorization can be found in the NAAS document.

5.5.10 Document Validation

Document validations are dataflow specific and Node specific. This section gives general recommendation on how to validate XML documents using DETs.

DETs identify types of information (data element and data groups) required or allowable for a particular type of data. DETs can take two forms: DTD or XML Schema. Verifying a document based on DTD is a very tedious, error-prone process (involving many string comparisons). This is because a DTD only defines the structure of the expected document, with no definitions about the internal data elements (such as data types, data ranges, max and mines, occurrences etc). Furthermore, because DTDs are not in XML format, it is difficult to use XML tools to automate the data validation process.

It is highly recommended that DETs be defined using XML schemas. Here are the reasons:

- Standard: XML schema is an open standard for XML validations, and it is more expressive than DTD.
- Quality: XML schema can not only be used to model DTD structure, but also data element definitions including data types, data ranges, char set and max/mins. With the precision of schemas, data quality and consistency can be much higher.
- Efficiency: With XML schema, there is no need to develop customized validation components for dataflows. Validations are automatic and supported by most of the popular XML parsers (parsing with validations).

In summary, an XML DTD will naturally require a lot more hand crafted validation because the DTD cannot incorporate definitions about the internal data elements (such as data types, data ranges, max and mins, occurrences etc), whereas an XML Schema can more fully specify those detailed definitions, and a validating XML parser can automatically use that fuller specification to detect errors.

The dataflow diagram in Figure 21 shows how DETs and XML schemas are used in a typical data exchange operation.

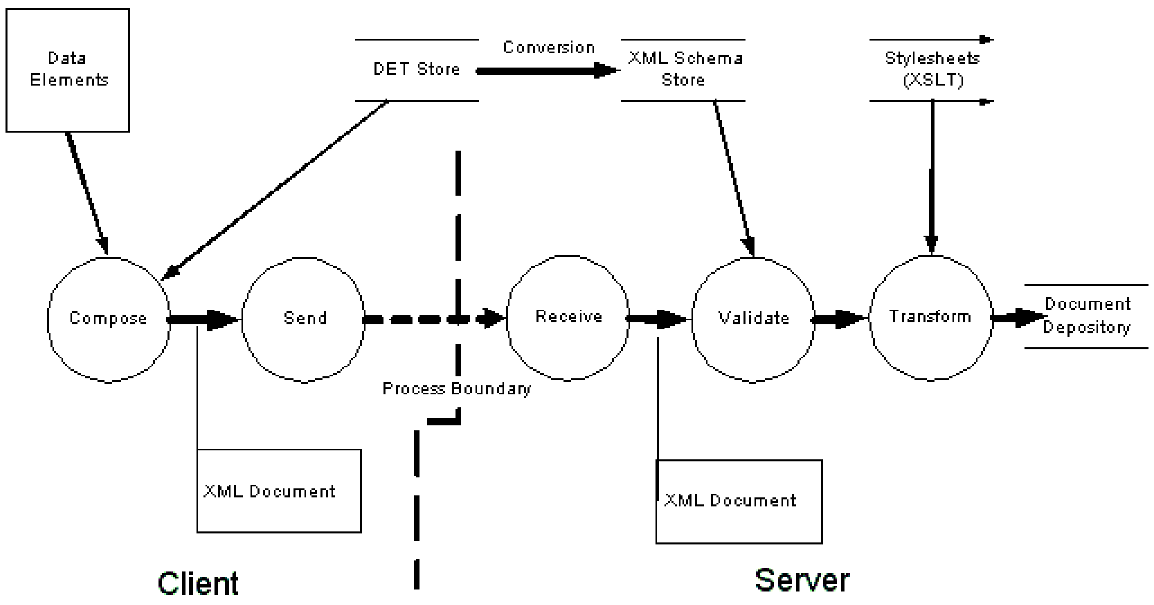


Figure 21 – Simplified document flow with DETs and XML schema

6.0 Describing Network Services

Ultimately, the exchange Network is envisioned as a dynamically expanding set of environmental information services. This dynamics will require a sophisticated and machine-readable process for the description of services so that clients can use them immediately.

Unfortunately, the technologies for this dynamic documentation, discovery and client generation are immature. The Network Node Specification V1.0 authors, therefore elected to establish, as the Exchange Network Protocol, this relatively static set of basic services, which could be easily implemented and used. Thus the WSDL file describing the entire Protocol is expected to be stable for the next 12-18 months. This has the great advantage that once a partner has established their Node, they can treat it as a “black box” which translates service requests into simple local procedures and parameters. For example, new queries can be supported simply by mapping the request to the appropriate stored procedures. There is no need to rebuild the Node itself.

Unfortunately, this decision also means that all of the lower level, exchange specific details of exchange business processes are locked into a Network specific proprietary human only readable format. That is, the documentation for the queries, downloads or solicit operations that a Node should (or does) perform are NOT expressed in XML. Instead they will be documented on the Network Website in human readable documents.

The Node 1.0 group considered an alternative approach, where each query, or solicit operation named would become part of the Protocol itself, and therefore documented in the Protocol WSDL file, but this would have meant that each partner’s Node (and its documenting WSDL file) would be in a constant state of flux as new services were added.

The payoff of this decision however is significant. The Network Exchange Protocol V1.0 WSDL file supplied with this document is the WSDL file for the Network. In most cases partners will need only edit a few parameters and can use the file to both generate and document their Nodes. This level of stability and predictability should accelerate implementation and position all partners to take advantage of upcoming improvements (especially WSDL/1.2).

In the interim, most partners will NOT need to develop custom WSDL files to describe the services on their Nodes. The Network conventions for how critical flow parameters (e.g. file names for download/submit, and the request/source and parameter values for query/solicit methods) will be developed based on the early implementation experience of this Protocol.

Every Network Service can optionally be described electronically in a separate description file encoded using the Web Services Description Language (WSDL). The electronic description of the Web service can be used by advanced tools to automate the construction of the SOAP messages in the service interactions and business processes described above.

All Network transactions must have a WSDL file. The WSDL file must be accessible through the Network Registry and be referenced in the FCD. Table 6 contains all required WSDL tags and the details of Network use. Any WSDL file that does not contain all of the tags and/or conform to the Network use defined in does not conform to the Network Exchange Protocol V1.0 and is violating terms defined in the FCD governing the data exchange.

WSDL TAG	NETWORK USE	Supported Use Cases
<definitions>	WSDL root element	
<types>	Data type definitions	Container for XML schemas
<schema>	XML schema definitions	Contains data structure definitions and extensions
<message>	SOAP message definitions	
<portType>	Abstract port types	
<operations>	Assign messages to operations	
<input> <output>	Inbound and outbound message definitions	
<binding>	Message binding definitions	
<service>	Service definition	The element doesn't appear in abstract service definitions
<port>	Endpoint specifications	
<soap:address>	An endpoint URL for the service	

Table 6 - Mandatory WSDL Tags

Table 7 contains all WSDL tags that are not required by the Network Exchange Protocol V1.0 but are supported by the Network. Use of these tags is optional and does not constitute a violation of the Network Exchange Protocol V1.0.

WSDL TAG	Network Use	Supported Use Cases
<documentation>	Provide online documentation of services and operations	
<dime:message>	Define layout of a DIME attachment	This really is an extension to WSDL
parameterOrder	This is an optional attribute for operation where it defined the order of parameters	

Table 7 - Optional WSDL Tags

6.1 WSDL Structure

The elements of the WSDL file provide information that will be used to generate the different parts of a SOAP message.

6.2 Types

The type can be either an element or a complex data type.

6.3 Messages

A message is a logical grouping of parts, each of which is either an element or a complex data type.

6.4 Operations

An operation is a logical grouping of messages that can be defined as either “input” to the Web service, “output” from the Web service, or a “fault” or error returned by the Web service. This is the basic information needed to generate the operational primitives that are the foundation of the Network service interactions.

There are four (4) types of operations:

1. One-Way
2. Request/Response
3. Notification
4. Solicit Response

6.5 Port Types

A port type ties input and output messages together as a request-response pair corresponding to a method invocation. An operation inside port type without output message indicates that it is a one-way operation.

6.6 Bindings

A binding defines the physical representation of messages on the wire and how they are transferred on the transport layer. In other words, the binding specifies how messages are serialized or de-serialized. A Network Node must support both SOAP-RPC and document/literal style encoding.

6.7 Services

A Service element describes a physical Web service,(i.e., which binding to use and where the service is hosted known) as the endpoint).

Note that service is an optional element in the WSDL specification. A WSDL file without a service element defines an abstract interface, which could be implemented by many service providers. In fact, all WSDL files listed in the UDDI registry must not contain any service definition.

6.8 Example

The following sample shows a WSDL file for the Authenticate service. Some namespaces are removed for clarity.

```
<definitions  name ='cdx'   targetNamespace = 'http://www.exchangenetwork.net/schema/v1.0/node.wsdl'
            xmlns:tns='http://www.exchangenetwork.net/schema/v1.0/node.wsdl'
            xmlns:xsd='http://www.w3.org/2001/XMLSchema' >
 <message name='Authenticate'>
  <part name='userId' type='xsd:string' value='cdx'/>
```

```
    <part name='credential' type='xsd:string' value='test'/>
    <part name='authType' type='xsd:string' value='password'/>
  </message>
  <portType name='NetworkNodePortType'>
    <operation name='Authenticate' parameterOrder='userId credential'>
      <documentation>User authentication method, must be called initially</documentation>
      <input message='tns:Authenticate' />
      <output message='tns:AuthenticateResponse' />
    </operation>
  </portType>
  <binding name='NetworkNodeBinding' type='tns:NetworkNodePortType' >
    <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
    <operation name='Authenticate' >
      <soap:operation soapAction='' />
      <input>
        <soap:body use='encoded' namespace='http://www.exchangenetwork.net/schema/v1.0/node.xsd'
encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </input>
      <output>
        <soap:body use='encoded' namespace='http://www.exchangenetwork.net/schema/v1.0/node.xsd'

encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </output>
    </operation>
  </binding>
  <service name='NetworkNode' >
    <port name='NetworkNodePortType' binding='tns:NetworkNodeBinding' >
      <soap:address location='http://epacdxnode.csc.com/xml/node.wsdl' />
    </port>
  </service>
</definitions>
```

7.0 Publishing and Discovering Network Services

All Web services must be registered in the Network Registry and be referenced in the FCD. The UDDI is the specification for describing, discovering and integrating Web services. It enables Network participants to both publish and find Web services.

The exchange Network will create and operate one private UDDI registry shared by all Network Nodes. The host of the UDDI service is called the UDDI operator.

One of the lessons learned from the public UDDI registries is that the quality of a registry determines its usefulness. Therefore, it is important to have a closely controlled, managed and maintained registry service for the Network exchange. The scale of the registry may be low, but the accuracy and precision must be high in order to have sound discovery and smooth integration.

7.1 UDDI Data Model

A UDDI registry has four (4) major entity types:

1. **businessEntity**: Describes a business or an organization that provides Web services.
2. **businessService**: Describes a set of services provided by a businessEntity.
3. **bindingTemplate**: Defines how services can be accessed. bindingTemplate provides the technical information needed by applications to bind and interact with the Web service.
4. **tModel**: Describe a technical model. It often contains an abstract definition of a Web service (Web Service Type).

All Nodes participating in the Network exchange must register as a business entity in the UDDI registry. There is a dependency between businessEntity and businessService: a businessService cannot exist without a provider, (i.e., a business).

7.2 Publishing Rules

The goal of the private UDDI registry is to create an accurate, consistent, dedicated registry for environmental information exchange. It is thus necessary to establish rules and guidance on who can publish, where to publish, and how to publish.

1. A service provider must be approved in order to register in the UDDI registry.
2. A participating Node can create business entities, business services and binding templates in the registry.
3. The UDDI operator must perform a Quality Assurance (QA) review on all newly created entities.
4. It is the responsibility of the Node to provide reliable Web services once registered.
5. The Node operator is responsible for creating a tModel, registering common interfaces (for instance, the Send interface, the Receive Interface, the Database interface and the Admin interface).
6. Authentication is required for all publishing operations.

When searching for Web services, one of the key pieces of information requesters are looking for is the WSDL file associated with the Web services. The WSDL file is registered as the overview URL of a tModel in the UDDI registry. Since a tModel represents a type of Web service, i.e., a common abstract interface, the WSDL file, pointed to by the overview URL, must not have any service definition (the <service> tag).

To register a Web service that complies with a tModel, the provider then creates a business service with a binding template pointing to the tModel. In other words, the tModel Instant of the service has the same tModel key as the tModel. This is how the Web service is associated with the tModel, and where the WSDL file can be located.

7.3 Inquiry Functions

- find_binding
- find_business
- find_relatedBusinesses
- find_service
- find_tModel
- get_bindingDetail
- get_businessDetail
- get_businessDetailExt
- get_serviceDetail
- get_tModelDetail

In a typical application scenario, to discover and to invoke a Web service dynamically, a requester uses the following invocation sequence:

1. Call find_service with a set of search criteria. This returns a list of Web services.
2. Choose the best one from the service list, and invoke the get_serviceDetail method. The bindingTemplate inside the service entity should have an accessPoint, which is where service requests should be sent.
3. Get the tModelKey from the bindingTemplate, and call the get_tModelDetail method. This provides the WSDL file associated with the service.
4. Invoke the Web service using the accessPoint and the service definitions (WSDL).

The procedure of searching a UDDI registry is further elaborated in Section 0 - *Accessing Service Information in UDDI.*

7.4 Publishing Functions

- add_publisherAssertions
- delete_binding
- delete_business
- delete_publisherAssertions
- delete_service
- delete_tModel
- discard_securityToken

- get_assertionStatusReport
- get_securityToken
- get_publisherAssertions
- get_registeredInfo
- save_binding
- save_business
- save_service
- save_tModel
- set_publisherAssertions

This set of functions is used to publish and update information contained in the UDDI registry. The publisher assertion APIs are for modeling complex business relationships, which are rarely used in the Network Exchange Protocol V1.0.

Note that all the operations are synchronous and atomic. The operation either failed completely or succeeded completely. For instance, there will never be partially saved business entities.

To protect the registry, users are required to login using the get_securityToken function before publishing any data in the registry.

7.5 UDDI Errors

UDDI Errors are presented as SOAP faults. In addition to the standard fault elements mandated by the SOAP specification, a UDDI fault message contains a dispositionReport in which registry-specific errors are included. The following sample shows the structure of a UDDI fault message:

```
<Envelope xmlns="http://schemas.xmlsoaporg.org/soap/envelope/">
   <Body>
     <Fault>
       <faultcode>Client</faultcode>
       <faultstring>Client Error</faultstring>
       <detail>
         <dispositionReport xmlns="urn:uddi-org:api_v3">
            <result errno="10500">
               <errInfo errCode="E_fatalError">The findQualifier
                  value passed is unrecognized</errInfo>
            </result>
         </dispositionReport>
       </detail>
     </Fault>
   </Body>
</Envelope>
```

The disposition report in this example contains an errno, an errCode, and an error description. Note that the errno (numeric code) and errCode (string error code) represent the same error in different forms.

UDDI error codes that apply to all UDDI API, common error codes, are listed below:

- E_securityTokenExpired

- E_securityTokenRequired
- E_busy
- E_fatalError
- E_requestTimeout
- E_unrecognizedVersion
- E_unsupported

8.0 Interacting with Network Registries & Repositories

UDDI is a directory or registry, not a depository or repository. This means that UDDI, unlike ebXML, does not physically store WSDL or XML schema files. So, in order to operate properly and efficiently, there needs to be external storage for all WSDL and DET files.

All referenced schema and WSDL files can either be stored in a virtual directory, or distributed to their owners. There are different strategies to reduce inconsistencies and maintenance cost. All the schemas must be referable using URIs under the requirements of XML namespaces as well as the import/export operations.

Following W3C conventions, one possible approach might look like the following:

For the WSDL files:

`http://www.exchangenetwork.net/schema/v1.0/node.wsdl`

For the XML schema files for Toxic Release Inventory (TRI) data:

`http://www.exchangenetwork.net/schema/v1.0/tri.xsd`

There would then need to be a set of search and retrieve functions that would be implemented against the Network repositories.

8.1 Accessing Service Information in UDDI

There are four key data elements inside UDDI: Companies, Services, Binding Templates and tModels. Services on the Network, unlike those in public UDDI registries, are provided primarily by state Nodes and the EPA Node. Each service is assigned a unique key at the time of creation. It is easy to retrieve service details given the service key as shown by the following UDDI request message:

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
      <get_serviceDetail xmlns="urn:uddi-org:api" generic="1.0">
            <serviceKey>d5921160-…</serviceKey>
      </get_serviceDetail>
</Body>
</Envelope>
```

The response would be the service details including an access point, a service description and a tModel key. It, however, does not contain information about the WSDL file. To get the WSDL file, which is essential for invoking the service, one needs to get the tModel details using the obtained tModel key:

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
      <get_tModelDetail xmlns="urn:uddi-org:api" generic="1.0">
            <tModelKey>uuid:0e727db0-4…</tModelKey>
      </get_tModelDetail>
</Body>
```

```
</Envelope>
```

The overviewDoc, as shown below in the response, points to the location of the WSDL file:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <soap:Body>

    <tModelDetail generic="1.0" operator="EPA CDX Private" truncated="false"
xmlns="urn:uddi-org:api">

      <tModel tModelKey="uuid:0e727db0-3e14-…" operator="epa.gov/node/uddi"
authorizedName="0100001QS1">

        <name>State Node Interface 1</name>

        <description xml:lang="en">Notification Interface for State
Node</description>

        <overviewDoc>

          <description xml:lang="en">wsdl link</description>


<overviewURL>http://www.exchangenetwork.net/schema/v1.0/Node.wsdl</overviewUR
L>

        </overviewDoc>

        <categoryBag>

          <keyedReference tModelKey="uuid:…" keyName="uddi-org:types"
keyValue="wsdlSpec" />

        </categoryBag>

      </tModel>

    </tModelDetail>

  </soap:Body>

</soap:Envelope>
```

From a programming point of view, a Web service is completely described, and thus accessible, given the access point and its WSDL file.

8.2 Dynamic Invocation of Web Services

As described in previous sections, the UDDI registry is the key for building loosely coupled, dynamic Web service applications. When a Web service is moved to a different host, the service provider would update the service information in the UDDI and it would be available to all client applications immediately. The change will, in general, have very little impact on the client as well as other Network Nodes (peers) if a proper procedure is followed.

The procedure for invoking a Web service dynamically is outlined below:

1. Retrieve the access point and the WSDL location from the UDDI registry.
2. Download the WSDL file, normally through either HTTP or FTP.
3. Construct a request message based on definitions in the WSDL file.

4. Send the request message to the access point.

5. Process the response.

Note that if the request message is constructed at run-time, the Network will achieve maximum flexibility. It can even accommodate interface changes, such as redefining parameter types or adding new parameters.

Maximum flexibility is obtained at the cost of performance. As can be seen from the procedure above, several Internet connections have to be established before actually invoking a Web method. Since WSDL files are relatively stable, the recommended approach is to cache the files locally for subsequent invocations, and to refresh the files when a Network error occurs. The procedure is revised as follows:

1. If there is a local cache of the WSDL file, go to step 4.
2. Retrieve the access point and the WSDL location from the UDDI registry.
3. Download the WSDL file, normally through either HTTP or FTP, and save a local copy for future use.
4. Construct a request message based on definitions in the WSDL file.
5. Send the request message to the access point.
6. If the response status is a Network error and the cache is old, go back to step 2.
7. Process response messages.

This approach, known as one of the best practices in Web services and UDDI integration, allows requesters to consult the UDDI registry only when necessary, which reduces the load on the UDDI server and boosts performance of the Node services.

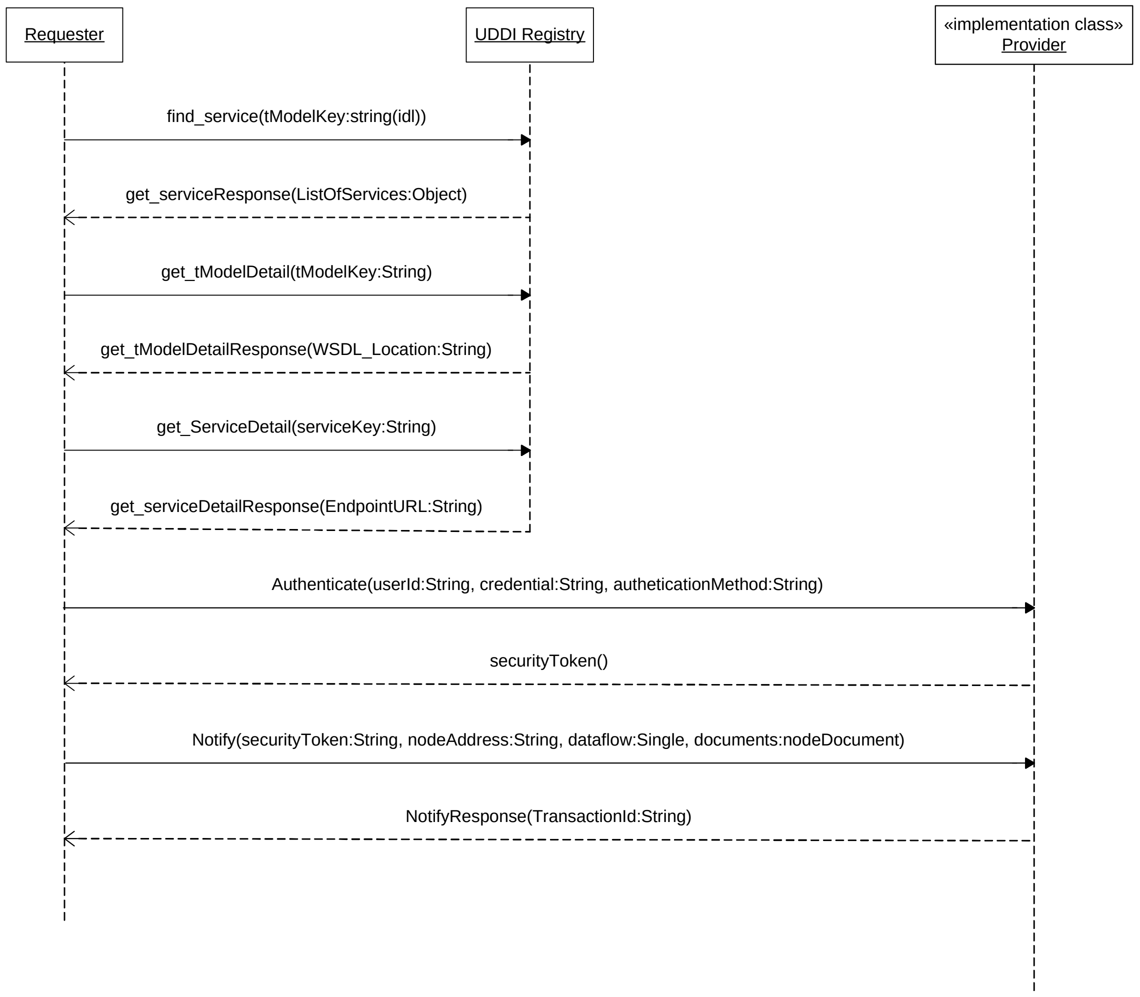
8.3 Using UDDI for Broadcasting

Broadcasting allows a Network Node to send messages to multiple recipients (listeners.) The broadcaster Node sets up an abstract interface (e.g., status notification method) as a tModel in the UDDI registry. Network Nodes that would like to join the broadcast, register a Web service in UDDI that supports the required interface methods (e.g., a Web service that supports the Notify method.) This allows the broadcaster to find all listeners using a simple UDDI request similar to the following:

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">

<Body>

      <find_service businessKey="*" xmlns="urn:uddi-org:api" generic="1.0"
maxRows="100">

      <findQualifiers></findQualifiers>

            <name>%%</name>

            <tModelBag>

                  <tModelKey>UUID:0E7F7DB0-3E14-11D5-98BF-
002035229C64</tModelKey>

            </tModelBag>

      </find_service>
```

```
</Body>
</Envelope>
```

Suppose UUID:0E7F7DB0-3E14-11D5-98BF-002035229C64 is a tModelKey assigned to an intrusion detection interface. Then the above request returns all Network Nodes that are willing to be notified when the event occurs. Armed with a list of endpoints and the WSDL file, it is not difficult for the broadcaster to call all the listeners concurrently or sequentially. Figure 20 shows a sequence of operations that involves three parties.



Requester
UDDI Registry
«implementation class»
Provider
find_service(tModelKey:string(idl))
get_serviceResponse(ListOfServices:Object)
get_tModelDetail(tModelKey:String)
get_tModelDetailResponse(WSDL_Location:String)
get_ServiceDetail(serviceKey:String)
get_serviceDetailResponse(EndpointURL:String)
Authenticate(userId:String, credential:String, autheticationMethod:String)
securityToken()
Notify(securityToken:String, nodeAddress:String, dataflow:Single, documents:nodeDocument)
NotifyResponse(TransactionId:String)

Figure 20 - Broadcast Operation Using UDDI Registry

9.0 Error Handling

During the course of the execution of the various interactions and scenarios outlined above, errors may occur. An error in this context can be either a malfunction of one or more of the components participating in a transaction, or it can be a violation of one or more of the rules of the Protocol. The errors can occur at any level of the Network Web service Protocol stack.

Validation of message payloads against the XML schema must be performed immediately upon receipt of any message. It is the responsibility of service providers to verify SOAP requests, and send SOAP fault message to the requester.

SOAP 1.1 classifies faults into four classes: VersionMismatch, MustUnderstand, Server and Client. Most of the SOAP implementations handle the first two classes of errors automatically, so there are really only two major faults categories left. The SOAP-ENV:Client faults indicate that the message was incorrectly formed or did not contain the appropriate information in order to succeed. The requester should, in general, retry the request with corrections. The SOAP-ENV:Server faults, on the other hand, indicate that the service provider is unable to fulfill its obligations due to some internal difficulties. The requester should retry sometime later.

10.0 Security

A major requirement of the Protocol is that it facilitates and participates in establishing and maintaining secure communications. There are three parts to security: prevention, detection and response. The Network is responsible for providing for prevention through the mechanisms discussed below. The individual trading partners are responsible for providing for detection of and response to security breaches within their purview.

SOAP and Web services have proven to be a powerful framework for creating distributed computing Networks and for conducting wide area of information exchanges. They are, at the same time, a big challenge to information security. Due to the nature of SOAP transports, which are based on public information exchange Protocols for the Internet, Web services open the doors for direct and indirect attacks from hackers and enemies alike. Web services without security measures are very vulnerable.

This section discusses available technologies for securing Web services, and addresses security issues from three major areas: Authentication/Authorization, Confidentiality and Integrity.

10.1 Applicable Security Protocols

10.1.1 HTTP Security

HTTP offers some basic authentication services on the transport level. The HTTP Specification (RFC 2616 and RFC 2617) defines an authentication mechanism known as "Basic" authentication. A client is challenged to provide identification information if authentication is required. The client then sends user name and password in the Authentication header. At this time, the user credentials can be passed to the Web service for verification.

A more secure but less popular HTTP authentication scheme is the Digest Auth. Instead of sending user passwords through the wire, Digest Auth sends an MD5 hash (a one-way hash algorithm that produces a "fingerprint" of the given data) of the user name, password, and other security elements to the server.

HTTP authentication schemes are considered weak in term of confidentiality. Information exchanges between client and server are clear text, which are subject to attacks. Therefore, HTTP authentication is not recommended for securing Node operations.

10.1.2 SSL

Basic Network security will be provided through SSL. Since its introduction in 1995, SSL has become the de facto way to secure communications between HTTP requesters and HTTP servers. It provides adequate confidentiality at the session layer, where data is encrypted by the senders and decrypted by the recipient using public key technologies. SSL, however, does not always provide a suitable method of authentication. Unlike B2C applications where the identity of a service provider is the main concern (client-side risk), client identities become the focus for Web services (server-side risk). Client side authentication through SSL, although possible, is always questionable due to the complexity of certificate management and the relatively high cost.

10.1.3 PKI

Public-key infrastructure (PKI) is the combination of software, encryption technologies, and services that verify and authenticate the validity of each party. PKIs integrate digital certificates, public-key cryptography, and certificate authorities into a Network security architecture.

While PKI in theory provides an effective, robust means of securing electronic communications and transactions, deploying and managing the technology remains a daunting challenge to many organizations, especially in a large-scale deployment.

10.2 Security Levels

There will be four levels of security supported by the Network Exchange Protocol V1.0. All message structures will incorporate (be surrounded by and encoded in) the various security Protocols associated with that security level.

10.2.1 Public Access

Public information that requires no authentication or certification of integrity.

10.2.2 SSL with Client Authentication

Information that requires some additional level of authentication and a higher level of integrity protection. It is protected through SSL plus application level client authentication (username and PIN). The Network Exchange Protocol V1.0 requires that this level of security must be implemented by all Nodes participating in the Network information exchange.

10.2.3 SSL with Dual-authentication

Information requires bi-directional authentication and a higher level of confidentiality. It is often protected using SSL with dual authentication. SSL with dual-authentication will be required depending on the dataflow, but is not mandatory for all Network transactions.

10.2.4 Digital Signature

Information requires non-repudiation and integrity protection in addition to privacy and authentication. Digital signature may be required by some dataflows. When required, it is strongly recommended that XML-Signature be used for digitally signing the documents, and the signature be inserted into the SOAP header part of the message under such situations. Methods for implementing XML-Signature will almost certainly be a part of future Protocol revisions.

10.3 Authentication and Authorization

All operations, except NodePing, in the Network Exchange Protocol V1.0 are restricted to registered users only. The restriction requires a user to be authenticated successfully before any other operations can be conducted.

Authentication is a process of establishing trust, (i.e., who the remote party is and what kind of privilege it has). Authorization relies on a good authentication scheme to protect Network resources.

Authentication is also necessary for establishing security policies based on users or user groups. It is also important for creating trusted relationships among Network Nodes (trusted peer relationship) so that highly confidential message exchanges, such as intrusion notifications, are possible between peers.

Authorization is a process of establishing entitlement of a subject. A user, although authenticated, may not be allowed to access certain Network services based on a security policy. Given the authenticated user identity (the subject) and the security policy of a Network resource (the object), a Network Node can determine whether or not to grant access. Authorization typically is a more complicate process than authentication, it is discussed further in the Network Node Security Guideline and Recommendations document.

To gain access to Web services provided by Network Nodes, a user must first send an Authenticate message similar to the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
…>

<SOAP-ENV:Body>

      <mns:Authenticate
      xmlns:mns="http://www.exchangenetwork.net/schema/v1.0/node.xsd">

            <userId xsi:type="xsd:string">JohnDoe</userId>

            <credential xsi:type="xsd:string">T34ngPRN2345INt</credential>

      </mns:Authenticate>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Note that namespaces are omitted for clarity. The authenticate message contains two elements:

- userId: User Id
- credential: User credential

Where the credential can be a password, a secure key value, or even a digital fingerprint, issued by the Network Node operator (service provider). Upon a successful authentication, the Network Node returns a securityToken, otherwise known as the digital ticket that will expire after a predefined time period. The user then includes the securityToken in all subsequent request messages as a proof of identity.

A SOAP Fault message, Unknown User, is returned if authentication fails.

A securityToken is an opaque string that is meaningful only to its issuer. It is usually an encrypted string that contains information useful for the validation of the ticket, and incomprehensible to its holder. To prevent replay attack, it must contain a timestamp so that token expiration can be enforced.

A simple securityToken may contain the requester's IP address, the user Id or profile name, a session ID for state tracking, and a timestamp for expiration. The result string is then encrypted and encoded using some secret algorithms.

A properly constructed securityToken can be highly secure. The issuer may validate the requester, the requester's machine and the timestamp. A stolen ticket has to be used on the same host within a very limited time window in order to cause a security breach.

This authentication process is based on the assumption that user registration and authentication are system specific, and beyond the scope of this document.

10.4 Central and Federated Authentications

A securityToken may, ideally, be issued through a central authentication server or a central security management (CSM) service. This approach, based on the NAAS is proposed by the Node 1.0 group for initial Network flows. It is certain to evolve in future Protocol releases.

This model has numerous advantages in several areas:

1. **Simplified Implementation**: Using CSM, state Nodes can simply delegate all security related tasks to the CSM service. For instance, the implementation of Authentication method and validation of authentication token become simple SOAP service requests.
2. **Enhanced Security**: With central security services, security risks shift from distributed Nodes to one CSM system. Defending the Network services is much easier from one point than from many points.
3. **Cost Effectiveness**: Security systems and related products are costly. A CSM service can dramatically reduce acquisitions of such product at state Nodes.
4. **Highly Extensible**: Upgrading security system to new technologies, e.g., from username/password to a PKI-based authentication using certificates, can be done relatively easily with the CSM system.
5. **Single Sign-On (SSO)**: Since authentication and validation take place at a single Node, the securityToken issued by CSM is applicable to all Nodes in the Network.
6. **Security Monitoring**: With CSM, it is possible to monitor all activities of the overall Network from a single location. This is essential for intrusion detection and vulnerability management.

The tokens issued by CSM are recognized and honored by all participating Network Nodes in a trusted relationship. A central authentication server facilitates single sign-on. Users need only register or login once in order to access services provided by all Network Nodes.

In a federated authentication scheme, however, each Node owns and manages a set of user identities locally and each Node is authorized to issue securityTokens. The securityTokens are recognized and honored by other Nodes in a trusted group. A federated authentication scheme is a distributed authentication system where Network Nodes are autonomous in that they have authoritative control over user identities registered at their site.

Single sign-on (SSO) can be achieved relatively easily in a centralized authentication environment. Figure 21 shows a simplified single sign-on configuration.

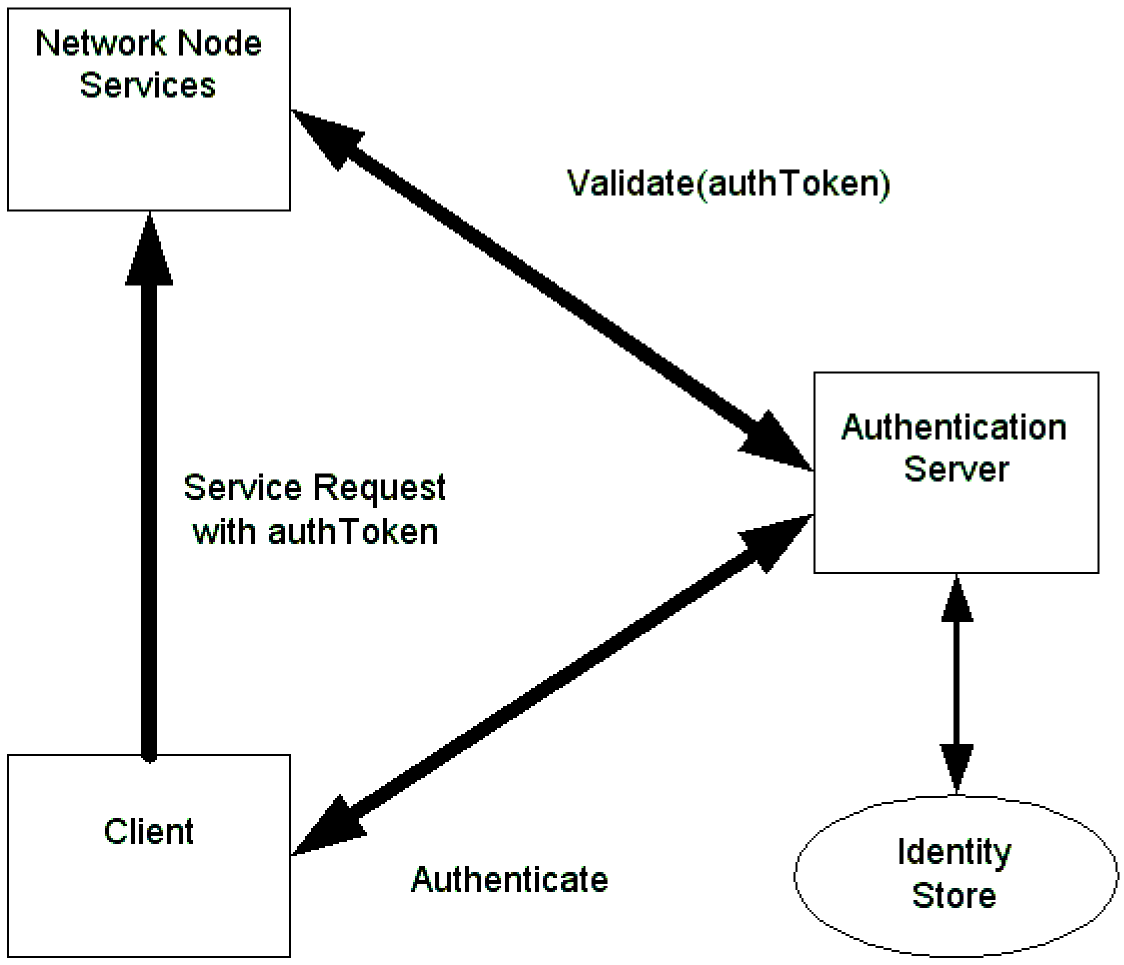


Figure 21 – Single Sign on Configuration

The process of SSO is outlined below:

1. The client sends a name and credential to the authentication server. The server returns a securityToken when successful.
2. The client then invokes a remote method on a Network Node, using the securityToken.
3. The Node sends the securityToken to the authentication sever for verification.
4. The authentication server checks the securityToken, and returns either a positive or negative answer.
5. The Node processes the request if the securityToken is valid.

Standards for establishing distributed trust relationships are currently under development. The Network Exchange Protocol V1.0 should incorporate such standards when available. For the time being, a simple solution would be use of a shared secret among Network peers. The authentication Node, Node A for instance, generates a session key using the secret, and then encrypts the securityToken using the session key. When the user presents the securityToken to Node B, the Node can generate the same session key using the given secret, and decrypt the token.

This discussion is provided to illustrate the extensibility of security approach described in the Protocol. It is expected, that in the first 12-18 months of Node implementations, that the NAAS approach will suffice for state/tribal/EPA flows. Partners will likely locally extend this approach for flows with regulated entities and others.

Note the following sections provide additional discussion of security issues but are not normative parts of the Network Exchange Protocol V1.0.

10.5 Message Confidentiality

Confidentiality is assured in most situations where messages are delivered through HTTPS transport. There are several situations, however, message may be compromised during transaction if not encrypted:

1. Use of transports such as SMTP or FTP.
2. Use of WS-Routing when messages travel over intermediaries.

It is strongly recommended that messages be encrypted using XML-Encryption under such application scenarios.

10.6 Message Integrity and Non-repudiation

SOAP message integrity can be protected using digital signatures, which assures that contents of a document were not tampered with during transition. Contrary to the popular belief that digital signature offers more protection than encryption, signature and encryption are actually integral parts of one thing: information security. Encryption only hides contents of a document; the contents can still be altered during transition. On the other hand, a digitally signed document without encryption is similar to sending an open letter without sealing it.

Another very important aspect of digital signature is non-repudiation. Some documents may require a digital signature to be considered valid by some dataflows from a legal point of view. Digital signature is no longer an optional feature in such situations.

The WS-Security specification, proposed by IBM and Microsoft, defines a set of processes and rules that applications must follow in order to be compliant and interoperable. It is desirable that the SOAP stack provider supplies an implementation of WS-Security as part of the SOAP toolkit.

For messages with attachments, calculation of digest should include all attached files. In other words, both the SOAP main message part and attachments should be protected by signing a combined digest of all parts.

An alternative approach is to generate a signature for each individual part, body and attachments, and insert multiple signatures in the SOAP message header. The approach adds extra processing in the SOAP header, but allows more flexible signature verification. Signatures, when present in a SOAP header, must have the mustUnderstand attribute set to **true**. Validation of signatures is **mandatory** on the receiver end.

The following SOAP header shows a dynamically generated digital signature:

```
<SOAP-ENV:Header>

      <SOAP-SEC:Signature xmlns:SOAP-
      SEC="http://schemas.xmlsoap.org/soap/security/2000-12" SOAP-
      ENV:mustUnderstand="1">
```

```
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">

              <SignedInfo>

                    <CanonicalizationMethod
              Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>

                    <SignatureMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>

                    <Reference URI="#Body">

                    <Transforms><Transform
                    Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
                    20010315"/></Transforms>

                    <DigestMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><Digest
                    Value>9r1eQL2syybnZXfx5wOECvl5nrs=

                    </DigestValue></Reference>

              </SignedInfo>

              <SignatureValue>MIIHbAYJKoZIhvcNAQcCoIIHXTCCB1kCAQExCzAJBgUrDgMCG
              gUAMIIByQYJKoZI

              jUVNX7rDA=

              </SignatureValue>

              <KeyInfo>

                    <KeyName>soapclient.com</KeyName>

                    <KeyValue>BgIAAACkAABSU0ExAAQAAAEAAQBHednVT1COLGAohJZqB8R1q
                    RUptRQbpWRhSZKG

                    GMmTU3s5m5TNe4iY4oP1/NxrjXCE7PjRX062y7mAKdkj55FcvDMhTcVLF5O
                    5xJTO

                    SVY5j8tcVpkTFKFKS3UXcJ1nyx+9UvwzGNzhKMgF8GIDHT58ZGz3yjbzb3V
                    mwmmW

                    0cdJvw==

                    </KeyValue>

              </KeyInfo>

        </Signature>

        </SOAP-SEC:Signature>

  </SOAP-ENV:Header>
```

The signature value is truncated for clarity. In this digital signature, the signer provided not only a signature value encrypted using a private key, but also a public key (in the KeyValue element) for decrypting the signature. The document is thus self-contained, verifiable by anyone who knows how to process a signature.

11.0 References

1. Network Exchange Functional Specification, Prepared for EPA by CSC, March 14, 2003.
2. Advanced SOAP for Web Development, Dan Livingston, Copyright 2002, Prentice Hall PTR, Upper Saddle River, NJ, 07458.
3. Web Services Essentials, Ethan Cerami, Copyright 2002, O'Reilly & Associates, Sebastopol, CA, 95472.
4. Programming Web Services with SOAP, James Snell, Doug Tidwell, Paul Kulchenko, Copyright 2002, O'Reilly & Associates, Sebastopol, CA, 95472.
5. XML Boot Camp Training Manual, TRG/Node 1.0 XML BOOT CAMP, June 10-11, 2002, SAIC, LMI, enfoTech, Philadelphia, PA.
6. Message Service Specification, Version 2.0 rev C, OASIS ebXML Messaging Services Technical Committee, February 2002.
7. W3C Node "Simple Object Access Protocol (SOAP) 1.1", May 22, 2000. (See http://www.w3.org/TR/2000/NOTE-SOAP-20000508/).
8. WS-Security, version 1.0. April 5, 2002.
9. W3C Working Draft "SOAP Version 1.2 Part 1: Messaging Framework", Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 26 June 2002 (See http://www.w3.org/TR/2002/WD-soap12-part1-20020626.)
10. W3C Working Draft "SOAP Version 1.2 Part 2: Adjuncts", Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 26 June 2002 (See http://www.w3.org/TR/2002/WD-soap12-part2-20020626.)
11. W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999. (See http://www.w3.org/TR/1999/REC-xml-names-19990114/.)
12. W3C Node "SOAP Messages with Attachments", John J. Barton.
13. Satish Thatte, Henrik Frystyk Nielsen, December 11, 2000.
14. Internet Draft " Direct Internet Message Encapsulation (DIME)", Henrik Frystyk Nielsen, Henry Sanders, Russell Butek, Simon Nash, June 17, 2002.
15. UDDI Version 3 Specification - http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, July 19, 2002.
16. W3C Node "Web Services Description Language (WSDL) 1.1", Erik Christensen, Francisco Curbera,Greg Meredith, Sanjiva Weerawarana, March 15, 2001 (See http://www.w3.org/TR/wsdl)

Attachment A
Network Operations Board
Network Decision Memorandum

NOB Memorandum Identifier:	**2005-03**
Issue Tracker Identifier:	
Memorandum Date:	**10/19/05**
Effective Date of Decision:	**10/19/05**

Topic: Valid Literal Representations, Use, and Interpretation of XML Schema Boolean Type Elements

Summary of Decision:

Non-XML conformant literal values for Boolean type elements have been used in instance files; these cause validation and processing errors and must be avoided.

This decision re-iterates the XML Schema standard allowed representations for Boolean elements, {true, false, 1, 0}[1] in instance files, and re-emphasizes the responsibility of instance files creators (i.e. data providers) to ensure that their instance files conform to the governing schema, including the proper representations for Boolean (and other) values.

Under XML Schema, legal literal representations for values of the Boolean datatype are {true, false, 1, 0}. However, across databases, and other interchange formats, there are many additional traditional representations for "Boolean" values; these include {TRUE, FALSE, T, F, True, False}. While valid in their respective domains, these representations are NOT valid values for XML schema elements of type Boolean. They must not be used. These non-conformant values should be detected by any validating XML parser, and should not be generated by any compliant XML generation tool. Post-generation validation is the most reliable way to ensure conformance of XML documents. While performance issues may preclude real-time validation of large XML files, data providers must at a minimum include validation of representative test documents as part of their deployment.

In addition, schema developers should ensure that use of Boolean data types is appropriate to the data being represented. Boolean variables, while providing very strong data typing, can cause confusion when applied to data for which a "third value" such as "U" or "Unknown" or "Undetermined" are allowed/expected. If values other than {true, false, 1, 0} are acceptable for a given XML element, a more appropriate data type must be chosen.

Affected Parties and Expectations:

Affected Parties	Expectations of Affected Parties
NTG	Publicize this decision Include guidance in upcoming revision of the Exchange Network Design Rules and Conventions

[1] See http://www.w3.org/TR/xmlschema-2/#boolean

Schema Developers	If values other than {true, false, 1, 0} are acceptable for a given XML element, developers must assign a data type other than Boolean to the element.
Data Providers	Data providers should validate all outbound XML documents against the XML schema to avoid data type mismatch errors.

Relevant Documents (e.g., guidance documents, decision memoranda, etc.):
n/a

History of Decision Memorandum Change:
In March 2006, the Network Technology Group (NTG) made minor revisions to this Decision Memorandum. These revisions served two purposes:

- To remove confusion around implementation of null in databases as compared to implementation of null in XML
- To clarify guidance for schema developers and data providers

Point of Contact:
Network Technology Group