FFI RAPPORT

EXPERIMENT REPORT: "SECURE SOA SUPPORTING NEC" - NATO CWID 2006

RASMUSSEN Rolf, EGGEN Anders, HADZIC Dinko,
HEDENSTAD Ole-Erik, HAAKSETH Raymond, LUND Ketil

FFI/RAPPORT-2006/02538

EXPERIMENT REPORT: "SECURE SOA SUPPORTING NEC" - NATO CWID 2006

RASMUSSEN Rolf, EGGEN Anders, HADZIC Dinko,
HEDENSTAD Ole-Erik, HAAKSETH Raymond,
LUND Ketil

FFI/RAPPORT-2006/02538

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
P O Box 25, NO-2027 Kjeller, Norway

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment

P O BOX 25
N0-2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-2006/02538	UNCLASSIFIED	39
1a) PROJECT REFERENCE	**2a) DECLASSIFICATION/DOWNGRADING SCHEDULE**	
FFI-II/898/912	-	

4) TITLE

EXPERIMENT REPORT: "SECURE SOA SUPPORTING NEC" - NATO CWID 2006

5) NAMES OF AUTHOR(S) IN FULL (surname first)

RASMUSSEN Rolf, EGGEN Anders, HADZIC Dinko, HEDENSTAD Ole-Erik, HAAKSETH Raymond, LUND Ketil

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS IN ENGLISH:	IN NORWEGIAN:
a) Network Based Defence	a) Nettverksbasert forsvar
b) Service Oriented Architecture	b) Tjenesteorientert arkitektur
c) Web Services	c) Webtjenester
d) Experimentation	d) Eksperimentering
e) Information security	e) Informasjonssikkerhet

THESAURUS REFERENCE:

8) ABSTRACT

During NATO Coalition Warrior Interoperability Demonstration (CWID) 2006 the Norwegian Defence Research Establishment (FFI) conducted the experiment “Secure SOA supporting NEC”. This document gives an overview of the experiment activities and results.

The technological goal for the experiment was to explore and demonstrate

- Dynamic Service Discovery, using a UDDI-based service registry
- Publish/Subscribe style information exchange based on Web Services
- End-to-end security, using labels and signatures to obtain object-level security
- Object-oriented use of the data model defined by MIP (OO/XML version of C2IEDM)

The experiment was a successful cooperation with the NATO research task group IST-061.

9) DATE	AUTHORIZED BY This page only	POSITION
2006-12-06	Vidar S. Andersen	Director

ISBN 978-82-464-1056-2

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (when data entered)

CONTENTS

EXPERIMENT REPORT: "SECURE SOA SUPPORTING NEC" - NATO CWID 2006

1 INTRODUCTION

This document describes the work performed by FFI-project 898 NBF Beslutningsstøtte to prepare and conduct the experiment “Secure SOA supporting NEC” (SecSOA in short) during NATO CWID 2006. A more detailed documentation, including technical implementation details regarding the Demonstrator, may be found in [10].

Section 2 is a brief description of the experiment background and context, including the Scenario, Test Cases and an overview of the FFI Demonstrator. In section 3 the four technological goals of the experiment are covered. For each goal there is a brief overview of the theory and a description of the experiment architecture supporting the goal.

Section 4 documents the results of the experiment with a subsection for each of the four technological goals, and section 5 is the conclusion.

2 OVERALL EXPERIMENT DESCRIPTION

The NATO Coalition Warrior Interoperability Demonstration (CWID) is an annual NATO Military Committee approved event designed to bring about continuous improvement in interoperability for the Alliance.

The NATO CWID programme focuses primarily on testing and improving the interoperability of NATO and national Command and Control (C2) systems. In addition to bilateral technical testing, NATO CWID provides a venue to conduct technical testing of fielded, developmental and experimental systems in the context of a coalition scenario.

Since 2004 the event has been arranged at Camp Jørstadmoen – primarily for three years, but the hosting has now been prolonged for 2007 and 2008.

The FFI-project ”NBF Beslutningsstøtte” has participated in the NATO Research Task Group IST-061, where it has been agreed to develop specifications and further implement experimental solutions for testing during NATO CWID 2006. The area of interest for the experiment has been four technical areas that will be further described in section 3:

- Dynamic Service Discovery
- Publish/Subscribe style information exchange
- End-to-end security
- Use of the Object-oriented version of the MIP [4] data model C2IEDM

The goal of the experiment was to prove that the combination of the four areas could be implemented, and demonstrate information exchange between nations based on the implementations. Underlying is the strong assumption that technology like this is essential to support Network Based Defence (NBD) or Network Enabled Capability (NEC), which is the NATO term for it.

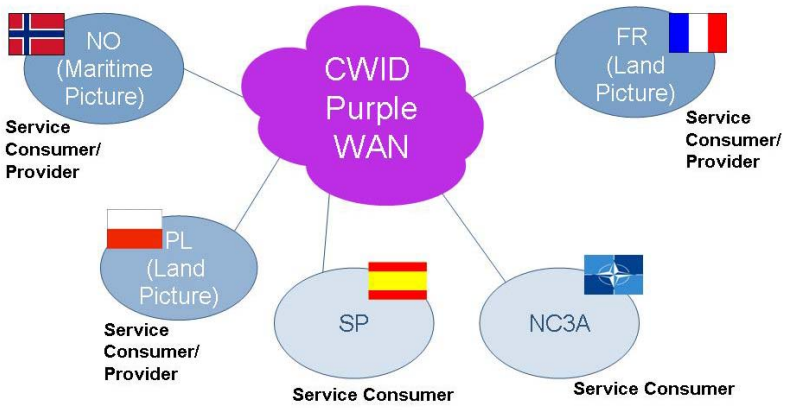

Figure 2.1 Participating Nations

From IST-061, the nations France, Poland and Norway were present at CWID. Germany and The Netherlands were not able to participate in the implementation. Spain and NC3A also joined the CWID experiment as illustrated in Figure 2.1.

The specification work was performed in the period from August 2005 to March 2006. On March 17 there was an email distribution to the research group of version 1.0 of the document "The NATO RTO/IST-061 Secure SOA Demonstrator Specification for CWID 2006" [6]. This document is the foundation for all the software development for this experiment.

Implementation work started early 2006 in parallel with finalization of the specifications. During May 2006, the development teams from each party were brought together in Oslo for preliminary system integration testing. Last week of May the teams moved to Camp Jørstadmoen and CWID, where the formal interoperability testing and demonstrations were performed.

2.1 Scenario

The goals for the experiment were primarily technological, but to be able to conduct a live demonstration of systems, we had to agree on a scenario. There was no need to comply with the official CWID scenario, in which the Interoperability Trials took place. So the decision was to define a simple scenario of our own.

The steps in the SecSOA scenario definition were as follows: An area in Southern England was selected as geographic area for the Demo. Further, the area was divided into squares that were allocated to each of the participating nations, as illustrated in Figure 2.2. In that way,

each nation could develop an independent set of objects and actions within their squares. When exchanging situational pictures, each nation would report their own area, causing no duplicates. And the total operational picture combined would come out pretty well.

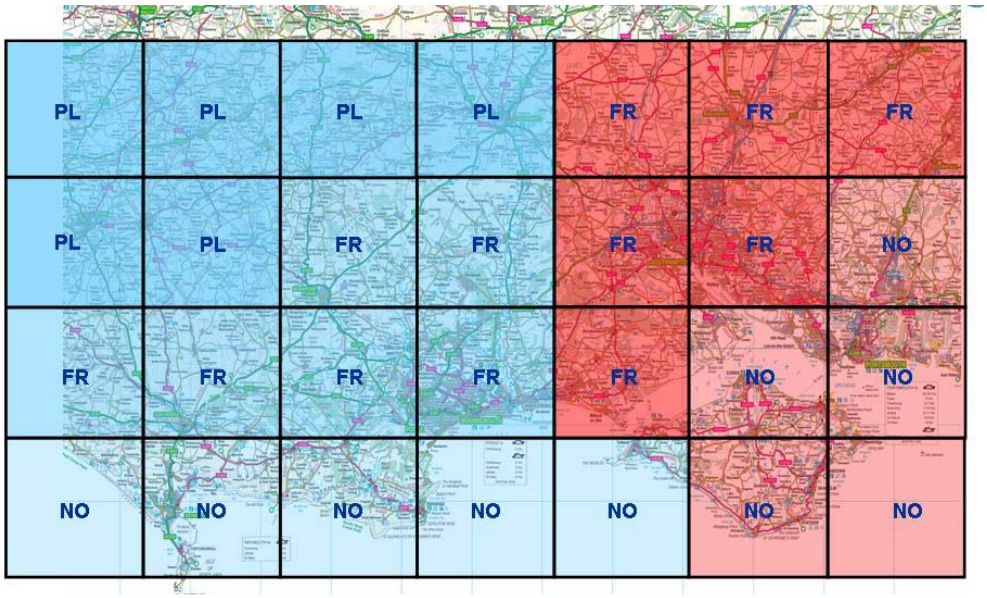

Figure 2.2 SecSOA Scenario Map

Norway was responsible of delivering the Maritime Picture, while France and Poland were to deliver Land Pictures. Norway was also to deliver MTI Tracks (MTI = Moving Target Indicator). All of these were to be delivered in a Publish/Subscribe manner.

For the demonstration, there was set up a special service called Sensor Request, to be delivered by Norway. The operational use of Sensor Request was that France, needing additional sensor coverage on an area (preplanned to be the Isle of Wight), would access this Norwegian service, requesting an area to be covered. In response, Norway would confirm that a sensor - being an Unmanned Aerial Vehicle (UAV) - would be launched, the time when sensor data would start coming, and the topic that the requester should subscribe to.

It should be pointed out that this hardly deserves the term ”operational scenario”. There were no coordinated operations, no ”master plan”, only a set of objects that may or may not move over time. Nevertheless, being extremely simple, the scenario turned out to be sufficient for the purpose of the experiment – to show that the technology for information exchange was working.

2.2 Test Cases

As part of the procedure for entering an Interoperability Demonstration into the NATO CWID management system, there has to be defined Test Cases. These are the criteria that the test results are to be evaluated against.

Traditionally at CWID, the Test Cases are numerous and quite detailed. ”*System A shall send <data> to System B*” may be one Test Case, and ”*System B shall send <data> to System A*” may be another. Given the coarse-grained scenario definitions that SecSOA was based on, and

the exploratory nature of the experiment, it was made an early decision to keep the Test Cases for SecSOA few and at a relatively high abstraction level.

For the Norwegian SecSOA we defined seven Test Cases. The numbering (TC#) refers to the identification in the official NATO CWID Test Case Tool. The Status column indicates the Test Case evaluation from Norwegian team.

TC#	Heading	Description	Status
615	Information delivery using Publish/Subscribe	Show that services are made available to others by publishing, and that efficient delivery of updates is achieved by subscribing to an information delivery service	Success
616	New services made ready for use	Show that a new instance of a well-known service interface, or a new service with a not previously defined data format, can be published and used	Partial success (50%) [1]
617	COI Cooperation	Show Net Centric cooperation between the C2 and ISR COIs using the object oriented MIP data model	Success
618	Enhanced end-to-end WS-Security	Show that all SOAP messages exchanged between nations are secured using PKI-based end-to-end object level security mechanisms	Success
619	Access control at the object level	Show that the information objects (WS-notifications or UDDI records) may be securely marked and that only users with the right security privileges are allowed to access/receive them	Partial success (90%) [2]
620	Distributed Security Management	Show that Certificates/user privileges can be issued or revoked, and evaluate the time needed till full effect among all nations involved	Success
621	Dynamic Service Replacement	Show that a broken service may be automatically replaced	Not tested

The experiment results are discussed in more detail in section 4.

2.3 Demonstrator overview

This section gives a brief overview of the Demonstrator. It was implemented by the FFI team with important contributions from Thales Norway, especially on the security modules.

The implementation builds upon the Picture Compilation Demonstrator used by FFI in former experiments, with substantial technical extensions as specified in [6].

[1] We did not test new data formats
[2] UDDI (see section 3.1.1) records were not secured due to lack of functionality in the implementation

The FFI demonstrator is a distributed system consisting of several loosely coupled modules deployed across different physical machines. Figure 2.3 illustrates the deployment at CWID 2006, where the demonstrator consisted of the following parts:

- Simulation environment (VR-Forces and SensorSim2)
- A set of interacting Data Publishing Nodes (DPNs)
- Service registry
- Security Management servers

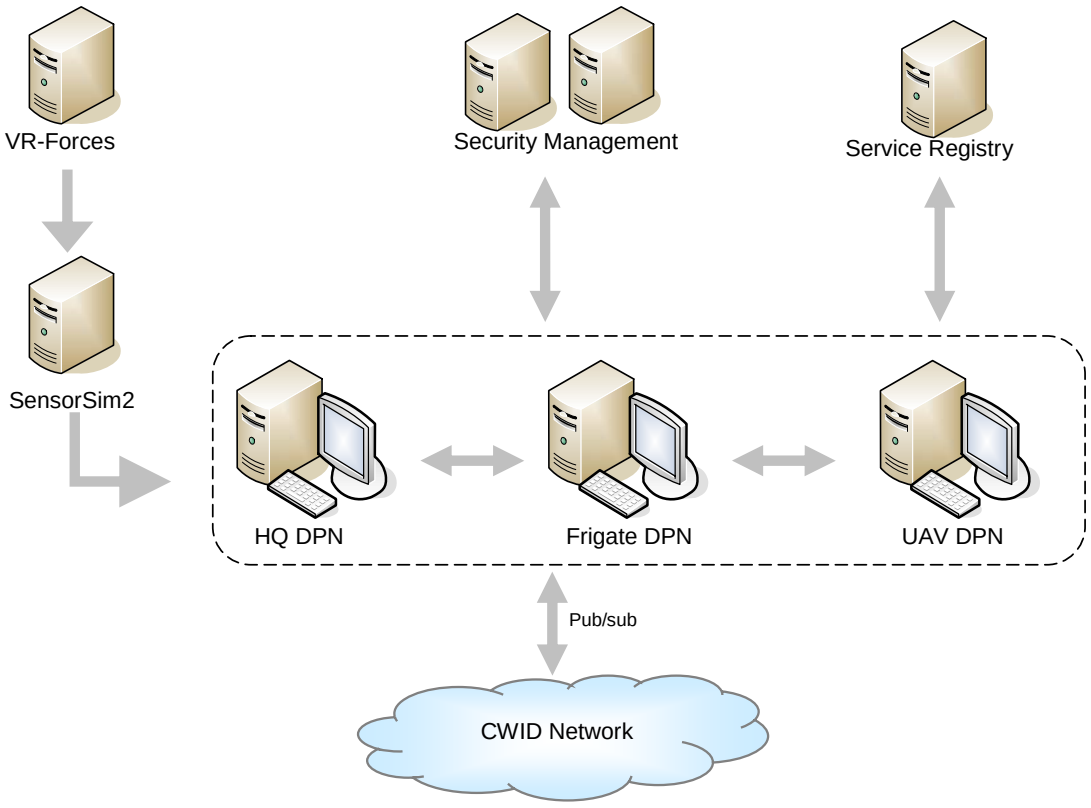

Figure 2.3 Demonstrator deployment CWID 2006

More details on the simulation environment may be found in [10]. The other parts are briefly described in the following.

Data Publishing Node

The DPN is a major component in the FFI Demonstrator. Each DPN represents a physical unit having a set of sensors attached to it, with an ability to communicate to other DPNs. Three national DPNs were used at CWID 2006:

- HQ DPN, which represents the national headquarters
- Frigate DPN, representing a frigate
- UAV DPN, representing an unmanned aerial vehicle (UAV)

Each DPN implements the Publish/Subscribe mechanism. Publish/subscribe allows a DPN to communicate with the network, either by acting as data producer or data consumer.

Seen from outside (other nations' view), the Demonstrator offers a set of services that can be accessed using the publish/subscribe interaction mechanism. Each service is offered at a single DPN. In addition, a DPN is responsible for building a Common Operational Picture (COP) based on data from different sources, and for information security.

Service Registry

The service registry was based on a commercial implementation of the UDDI v3.0 specification, provided by Systinet [13]. The registry allows the DPNs to register their services so they can be discovered by other DPNs. The FFI service registry was available to be used by all participating nations in the experiment.

Custom built and integrated with the Systinet registry were

- the UDDI abstraction layer outlined in section 3.1.3
- the registry client

Figure 2.4 shows a screenshot of the GUI of the registry client. The GUI provides information on UDDI entities and how they relate to each other on the left hand side, and a more detailed description of the chosen entity on the right hand side.

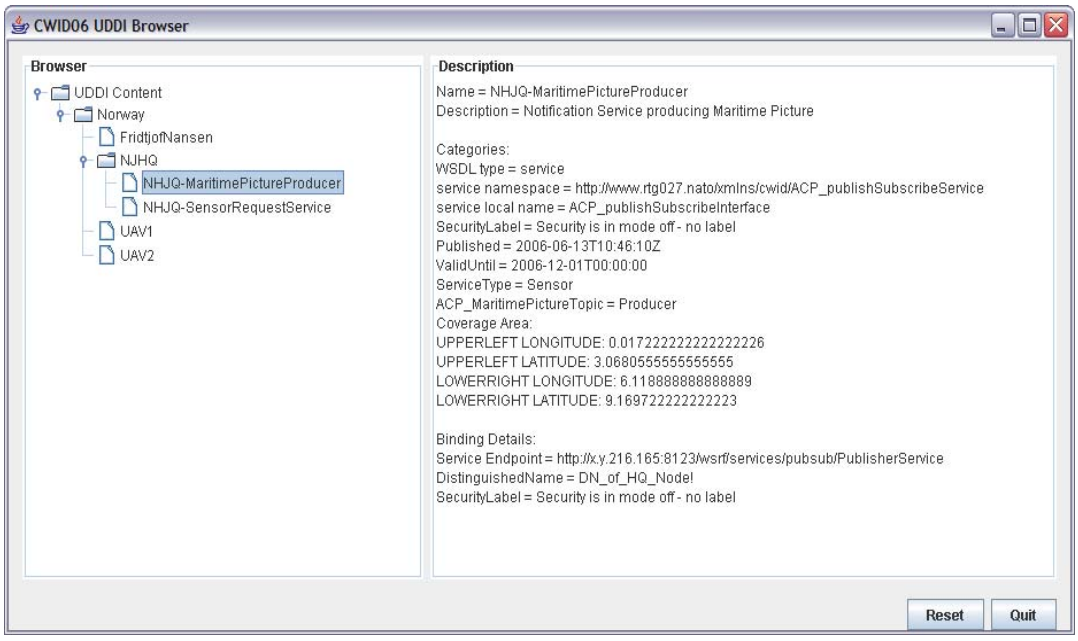

Figure 2.4 UDDI Browser

Security

For Public Key Infrastructure (PKI) we used the software product KeyOne from Safelayer [12]. A Lightweight Directory Access Protocol (LDAP, see also section 3.3.5) server contained security certificates. Each national LDAP server contained:

- Its own data stored under its branch (certificates and revocation lists)
- A copy of the data extracted from the other LDAP servers of the other nations

The security modules in the Demonstrator were developed by Thales Norway and made available to all participants in the experiment for integration into the national systems. The

Security Protection Component (SPC) was built into the DPN and the service registry. Being an integral part of these components, the SPC is not visible in Figure 2.3. The important role of the SPC is more properly illustrated in Figure 3.9.

SensorRequest

The SensorRequest service was an important part of the scenario. SensorRequest allows an entity requiring intelligence support to ask another entity for sensor coverage within a specified geographic area. Such a request is shown in Figure 2.5. Based on the request information, the decision maker can either accept or reject the request.

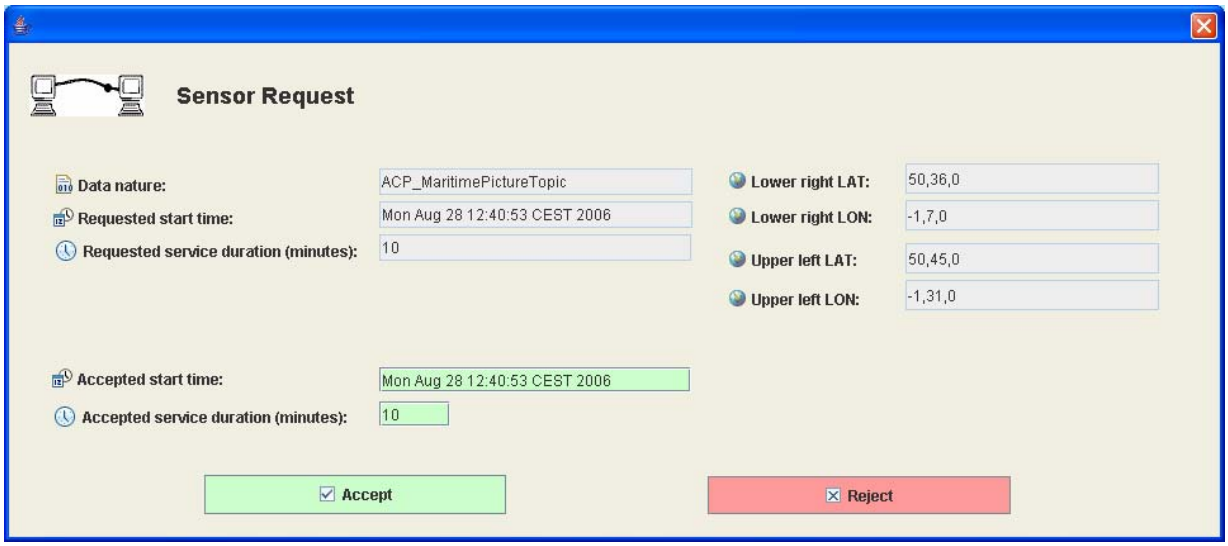

Figure 2.5 Sensor Request graphical interface

NORCCIS-II integration

Being an operational command and control information system, NORCCIS-II provides professional functionality for visualizing and handling tracks. To include NORCCIS-II in the demonstration, a Web Services based integration between the two systems was created:

- data received by the demonstrator was pushed to NORCCIS-II where it was presented
- NORCCIS-II could send messages to the demonstrator, which would relay these as notifications to all of its subscribers

The data exchange format was the object-oriented version of the MIP [4] data model C2IEDM.

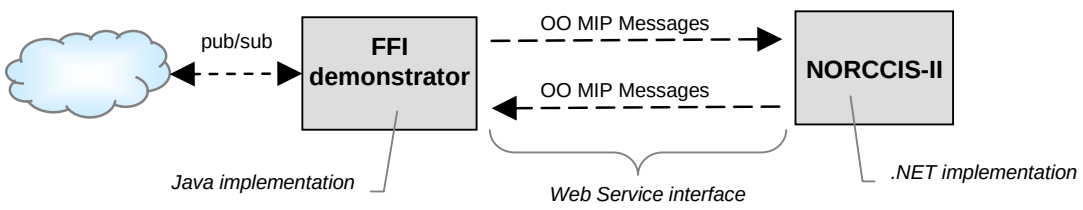

Figure 2.6 Two-way communication between the FFI demonstrator and NORCCIS-II

3 TECHNOLOGICAL GOALS

Before going into the details of the technological goals outlined in section 2, it is important to present the foundation for all these activities. The overarching theme for all areas of focus is the use of technologies for, and implementation of, Service Oriented Architecture (SOA). SOA is a powerful but simple architectural principle inspired by the way business is performed. Simplified an SOA consist of a Service Provider who offers its service, by publishing it in a registry, to Service Consumers. Service Consumers find these services by using the registry and is then able to bind to the Service Producer, as illustrated in Figure 3.1.

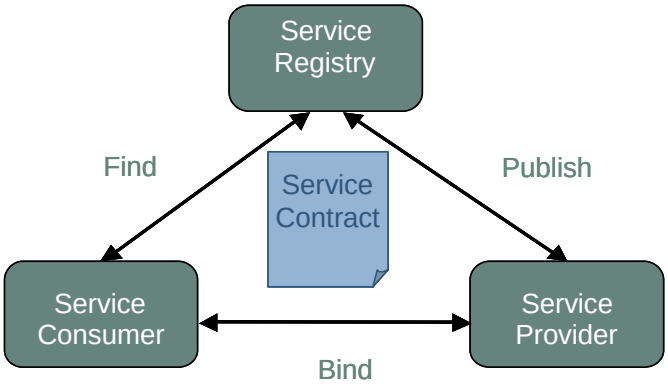

Figure 3.1 Service Oriented Architecture (SOA)

SOA may best be defined as a collection of services that communicate with each other. A service encapsulates standalone functionality which maybe delivered across a network. A service is well defined by a contract. The services can be combined to form the desired application or system.

Web Services is currently the preferred technology for implementing a Service Oriented Architecture. Web Services is essentially a set of XML based standards used to implement a SOA. As a consequence of the fact that Web Services is quickly becoming the de-facto standard to implement a SOA, we have based our work on these technologies.

3.1 Dynamic Service Discovery

3.1.1 Standards and specifications

In a Service Oriented Architecture (SOA) the ability to discover services during both design and run time is very important. This involves both finding and selecting services that match the current requirements from the client. Look up services have accompanied many technologies for distributed computing, e.g. JINI, Java RMI, JXTA and CORBA. For Web Services different alternatives exist for service discovery, the most common solution is the Universal Description, Discovery and Integration (UDDI) specification. Other alternatives include for instance the ebXML Registry [2] and Web Services Dynamic Discovery [14].

Service discovery may be separated into *design-time* and *run-time* discovery. *Design-time* discovery is utilized by client-software developers when designing and implementing client

software. By *run-time* discovery we mean service discovery performed during execution of a system. This may involve human intervention or be an autonomous process of searching, finding and choosing which service to use. Run-time discovery of services is often performed using a pre-known technical fingerprint of a service. This fingerprint may have been discovered by using design-time discovery. The experiment describe here involves both types of service discovery.

To enable dynamic service discovery in an NBD, a service registry is vital. Furthermore, the service registry should be able to provide support for environments ranging from static to highly dynamic. In contrast to the fairly stable service availability found in static environments, services and even networks may come and go in a non-deterministic fashion in a dynamic environment.

For our demonstration it was decided to use the UDDI specification. The argument for this was first of all that this is a specification that is in daily use and is perhaps the most used COTS specification for services registry and has strong vendor support. It has also been going through extended development to improve usability and performance through several versions. The latest version, UDDI v3.0 [7], was ratified as an OASIS standard in February 2005. Several advantages of this specification counted for using it versus the older UDDI v2.0, this includes e.g.; support for digital signatures, subscription API[3], support for multi-registry environments and better search API. Our main concern of using this specification was the lack of open source implementations. The solution was to use a commercial available product from Systinet, the Systinet Registry [13]. In addition, this satisfied our need for stability and performance at the registry.

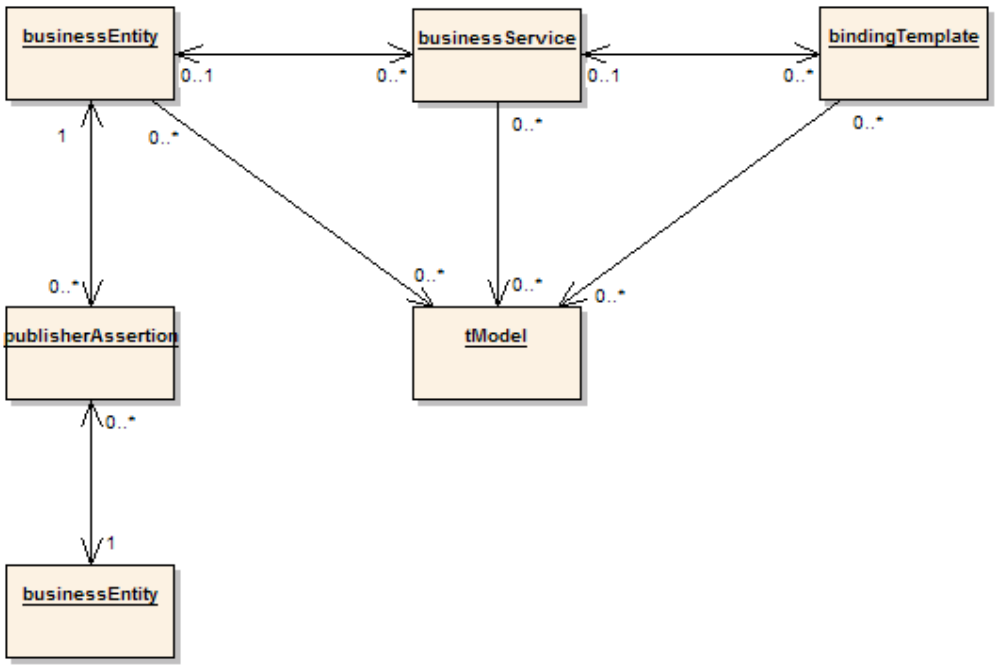

Figure 3.2 UDDI Data Model

[3] Application Programming Interface

The UDDI data model consists of four core entities; businessEntity, businessService, bindingTemplate and tModels, shown in Figure 3.2. The businessEntity entity is used to describe and represent providers of services like businesses and organizations. Relationship between businessEntities may also be established by using what is known as a publisherAssertion construct. This way one could visualize different types of relationships such as parent- and subsidiary companies, departmental structure within a company or different military units.

A businessEntity contains zero or more businessService entities. The businessService entity is used to describe services in a non-technical way. This description outlines the purpose of the service and may contain different metadata used for discovery of services. The businessService entity contains zero or more bindingTemplate entities. A bindingTemplate represents one individual implementation of the service described in the businessService. The information contained in a bindingTemplate is used by a client to bind to and interact with the service. The bindingTemplate is basically a collection of references to tModels, also known as Technical Models. tModels are used within UDDI to represent unique concepts or constructs like specifications, transport and protocols. The businessService entity with the referenced tModels together forms a technical fingerprint of the service. It is important to note that tModels are not confined to describing technical fingerprints. Other concepts like categorization schemes for businesses and services, identifier schemes and other might be expressed using tModels.

The decision made to use UDDI as the service registry left us in need to provide some additional functionality. First of all; the security mechanisms featured in the UDDI registry chosen is not inline with the security outlined for this experiment. Particularly the use of security label for access control is provided by neither the UDDI specification nor the implementation. For further details on the security mechanisms implemented please refer to [10].

The second additional functionality identified was service termination. A standard UDDI registry has no knowledge of how long an instance of a service will be valid. In order to avoid that the content becomes stale the registry is dependent on a graceful delete of services that are no longer valid. This is often not the case, and thus the registry content potentially becomes stale. To avoid this, all services to be registered in this experiment need to have an expiration time associated. The functionality to enforce this, including registration enforcement and deletion of expired services, must thus be implemented.

The third and last extra functionality identified is the need for extended search capabilities needed for military purposes. In specific there is a requirement to do inquiries for services within a specified geographic area, either for services physically placed within or having coverage of this area.

3.1.2 Use of the UDDI data model

In order to ease discovery of businesses and services we used an agreed upon set of metadata in combination with the UDDI data model presented above. The content of the defined metadata is taxonomies used to classify e.g. the service description, domain specific attributes of interest, service interface, transport protocols and message encodings. These taxonomies are

represented as canonical tModels. In common for all these tModels is that they are used to categorize or identify UDDI entities in order to ease the process of discovery. For this experiment 13 such tModels were defined. These can roughly be divided into two subgroups, namely those concerned with describing businessEntities and businessServices respectively.

To categorize businessEntities three additional tModels were defined. The first extension was the inclusion of an identification string which is used to identify the business uniquely within our own identifier system. The businesses are also categorized by what type or organization they represent. This is achieved by using a tModel named entity type. For this experiment three values for organizational units was identified; nation, asset and Community of Interest (COI). The third, and last, additional tModel used to categorize businessEntities is used to provide an improved and more accurate categorization of assets. The asset categorization tModel can be used to describe what kind of asset the described entity is, e.g. an UAV.

To categorize businessService entities, nine tModels were defined in addition to the predefined UDDI categorization tModels. Services can be categorized using the defined service taxonomy tModel. This describes what type of service this is, e.g. sensor. Included in our service categorization scheme is also the ability to describe geographical position and coverage area. The coverage area is described by using the coverage area canonical tModel in combination with the longitude and latitude tModels. The coverage area tModel is used to group together two references for longitude and two references for latitude, which together form a rectangle with upper left and lower right coordinates. In addition, the exact geographical position of the service may be described using the position tModel. It is important to note that this categorization only provides information on where this position can be obtained, e.g. a URL to a service providing this information, not the position itself. This is due to the fact that this is highly dynamic information thus not fitted for storing in the registry itself.

Another important categorization scheme included was the service termination policy. A service can, by using the published and valid until tModels, be categorized by when it was registered and when it is not valid longer. This can be used by clients to choose relevant services and by the registry itself to clean up and delete expired services. Categorization tModels are also used to represent security. This includes security labels and reference to security certificates represented by the LDAP distinguished name.

Support for WS-Notifications and registration of services supporting WS-Notification is not included in the UDDI v3.0 specification. In order to provide this support a categorization scheme for topics and topic spaces were introduced by using tModels. Each topic and topic space were themselves registered as tModels using these categorizations, and each service which produces data on the topic is linked with this. For more details on WS-Notification please refer to section 3.2.

In addition to the canonical tModels, we also defined tModels that are used to classify each service identified. These are specified according to the OASIS Technical Note describing how to publish WSDL files in UDDI [8].

3.1.3 Implementation of extra functionality

The extra functionality identified in section 3.1.1 was implemented using an abstraction layer in front of the UDDI server, see Figure 3.3. The abstraction layer functioned as an extra tier and implemented the necessary UDDI version 3 APIs, and no clients had direct contact with the UDDI registry. The choice of using this architecture with an additional tier was taken based on the fact we used a commercial UDDI registry from Systinet [13], and we did thus not have access to the source code. Since we didn't have access to the source code the only option left was the implementing the extra functionality outside the registry.

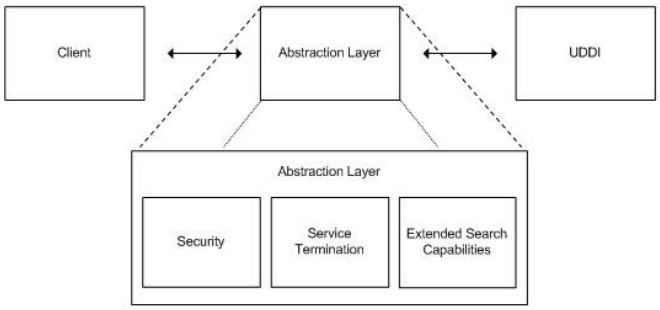

Figure 3.3 UDDI Registry with Abstraction Layer

The abstraction layer itself does not hold or store any information and use the information stored in the UDDI registry. As a consequence of this, the implementation of all the identified extra functionality essentially becomes filtering of return information from the registry. The exception from this is the security functionality which performs the initial check on message validity before messages are forwarded to the UDDI registry. Access control on UDDI objects are though performed by filtering the reply. Since no clients are in direct contact with the UDDI registry we can assume that all access to the registry is subject to these security checks. It must be emphasized that this is only a demonstrator setup and this is not a valid assumption when implementing this in a real life setting.

The extended search capabilities, in this demonstrator the geographical search, was also performed by using filtering of return values from the UDDI registry. As for the extra function of service termination we had two choices; either do active polling of the registry to discover expired services, or perform filtering of return information from the UDDI registry. Both alternatives have advantages and disadvantages, but in the end we chose alternative two, doing filtering. Before a reply from the UDDI registry is forwarded to the client it is filtered and all expired services is removed from the reply. The expired service is also marked for deletion from the registry. The main advantage of this approach is that we can assure that all information delivered is up to date. Doing this filtering may however reduce the response time of search inquiries.

In addition to the original UDDI v3.0 API functionality implemented by the abstraction layer, we have extended the publishing API by two methods. First, a method for publishing services

described by WSDL files and by the categorization information defined above. Second, a method for resetting the content, i.e. the services, of a businessEntity was implemented. This functionality was used for administration purposes during experimentation and demonstration.

3.2 Publish/Subscribe-style Information Exchange

3.2.1 Theory

Publish/subscribe, often abbreviated to pub/sub, is a well known communication pattern for event-driven, asynchronous communication. Publish/subscribe makes it possible to link together data producers and data consumers into loosely coupled, scalable and dynamic networks. We have chosen to rely on WS-Notification group of specifications from OASIS [9], which use Web Services to realize the publish/subscribe pattern. We have applied two WS-Notification specifications, namely WS-Topics [17] and WS-BaseNotification [15].

Using WS-Notification terminology, a service that publishes data at a specified Topic is called a *NotificationProducer*. The data format of each topic is well defined by an XML schema (XSD). A client, called a *NotificationConsumer*, first creates a subscription to the service. The client will subsequently receive notifications as they are produced by the NotificationProducer (see Figure 3.4).

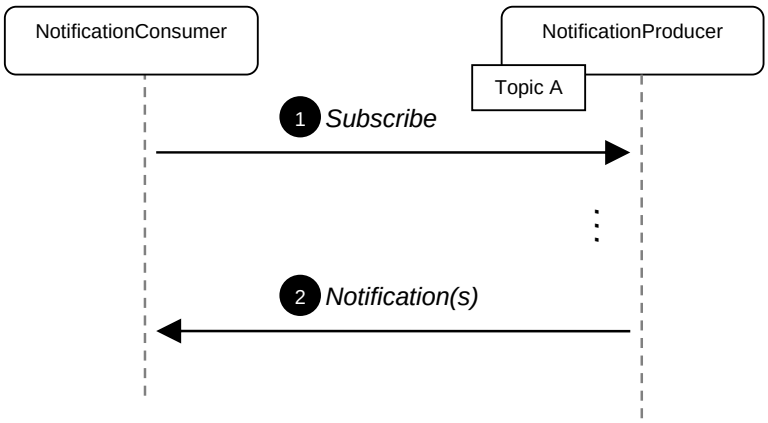

Figure 3.4 Publish/subscribe overview. NotificationConsumer creates a subscription to a Topic, and will subsequently receive notifications as they are produced

3.2.2 Architecture

Our goal was to utilize the WS-Notification family of specifications to realize efficient message distribution and dynamic communication management between national C2 systems.

By introducing the publish/subscribe pattern and WS-Notifications, we established a standardized way of communication and communication management (pausing, resuming, creating, destroying and renewing the subscriptions), which, in turn, is a significant advantage for interoperability. Each data publishing service could be accessed using the same interaction mechanism, regardless of how the national backend C2 system is implemented.

The publish/subscribe service architecture consists of a set of interconnected nodes called Data Publishing Nodes (DPN). Each participating nation developed and deployed at least one DPN in their national domain. The set of DPNs forms a NATO Data Publishing Network (NATO DPNet), illustrated in Figure 3.5.

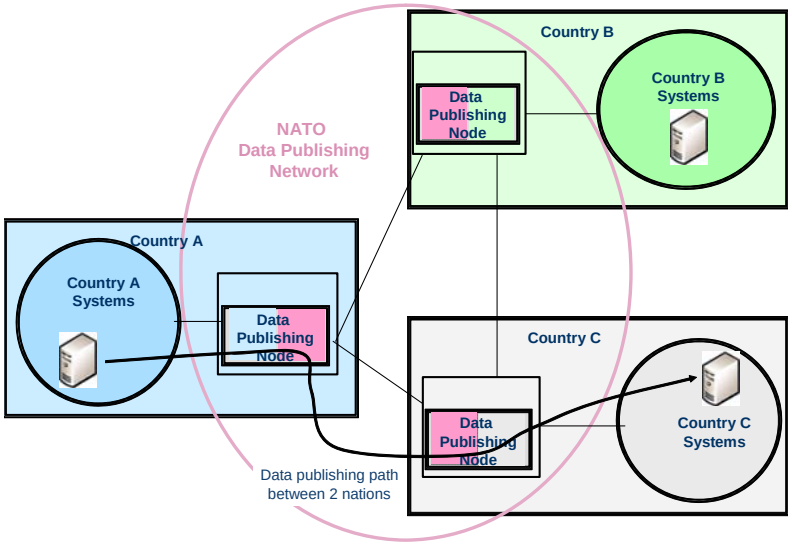

Figure 3.5 NATO Data Publishing Network (DPNet) consists of a set of national Data Publishing Nodes (DPNs)

A Data Publishing Node is the representative of a given nation on the NATO Data Publishing Network. However, a DPN is not itself a publisher or a subscriber. Rather, a DPN hosts and exposes to other nations a set of subscribers and publishers. Publishers and subscribers are logical entities structuring the public view of the nation's information production and consumption. It is each nation's choice, to expose only one publisher and one subscriber, or several of each (see Figure 3.6). How publishers and subscribers are mapped to or glued with the national systems is a national concern.

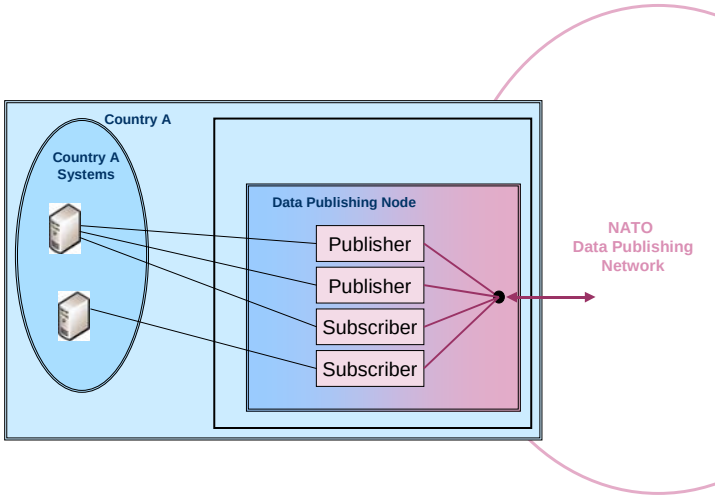

Figure 3.6 A DPN consisting of two publisher and two subscriber modules

The interoperability between nations is achieved by:

- The definition of common Topics and data format for each topic (data interoperability),
- The definition of a common publish-subscribe protocol between national publishers and subscribers (technical interoperability).

Three topics have been defined in our scenario:

- *ACP_MaritimePictureTopic*, providing maritime area tracks in C2IEDM format
- *ACP_LandPictureTopic*, providing land area tracks in C2IEDM format
- *ACP_MTITracksTopic*, providing tracks in MTI (Moving Target Indicator) format

In order to create a valid subscription to a NotificationProducer, clients need to provide the following set of parameters, which can by dynamically acquired from the UDDI registry:

- Address of the Publisher service endpoint, as defined by the WS-Addressing specification
- Topic definition, consisting of the name and the namespace of the topic

The data format of the notification messages is defined by an application-specific XML schema (XSD). The data format is implicitly given by the topic name, i.e. if the topic name is ACP_MaritimePictureTopic or ACP_LandPictureTopic then the data format is implicitly the reduced C2IEDM format, while the data format for the ACP_MTITracksTopic is MTI.

3.3 End-to-end Security

The increased information sharing in SOA may lead to increased vulnerability if security is not properly integrated. The situation of today is that separate networks protect information of different classification using physical, cryptographic and administrative separation. Introduction of security mechanisms which allows for dynamic and seamless exchange of information between units will be a challenge in NBD. IP level security will give confidentiality between systems, but will not prevent unauthorized access from within the systems or LANs. Computer Network Attacks (CNA) will focus on attacks behind the firewalls (crypto devices) within the LANs/Systems. Therefore, end-to-end security services are required in order to secure the information in the NBD systems and LANs.

Security is often thought of as a challenge with respect to NBD, making sharing of information difficult. In our experiment we have focused on application-level end-to-end security, which is highlighted as the long term goal in the NATO NEC Feasibility Study [5]. The use of end-to-end security solutions does not exclude additional use of traditional network and transport level security, but in this paper the latter will not be emphasized.

Use of the security technology described in this paper depends on adequate security policy and management procedures, which are assumed to be in place.

3.3.1 Specifications

All of the Web Services specifications are based on XML and most often the use of SOAP messages. Therefore XML general security specifications may be used for securing the different Web Services components.

Many specifications have been written for securing XML documents. Some of them have become standards. The major standardization organizations in this area are the W3C, OASIS and IETF. In addition Microsoft and IBM have developed the Web Services Security Road Map (further reference may be found in [11]), which describes a set of security specifications building on the OASIS WS Security standard [16].

What is missing in the wide variety of XML specifications and standards is an XML specification for security labelling of information objects. Security label specifications have earlier been developed for X.400 messaging (X.411 [11]) and SMTP (IETF S/MIME ESS [11]) and these may be used as a basis for the development of an XML Security Label specification.

The following bullets outline the security functionality that has been developed for SecSOA:

- All SOAP messages are attached a security label, encrypted and signed
- All advertisements in the service registry are attached security labels and signed before storage
- Before any notifications or UDDI records are sent to a requestor, her security privileges are checked against the security label of the information objects.
- A PKI and an LDAP Directory are used for providing the security infrastructure for exchange of certificates and certificate revocation lists.

The implementation uses a combination of several security mechanisms in order to achieve the goal of end-to-end security at the information object level. Each area is described in the following.

3.3.2 Basic security mechanisms

SOAP Security

All information exchange is done using SOAP messages. The security of the SOAP messages is based on the use of the OASIS WS-Security standard with extensions in order to include an XML Security Label. The OASIS WS-Security standard specifies how to extend the SOAP message header in order to achieve message integrity, confidentiality, authentication of originator and replay protection. The security label (and other important fields) is bound to the SOAP message by a digital signature. The content of the SOAP messages will be compressed, encrypted, labelled and signed before transmission. Upon arrival the security will be validated and the originator may be identified in order to see if the message comes from a reliable source.

Security Labels and User Security Privileges

A security label is attached to the information objects to be secured. This Security Label gives flexibility in marking the information, and is an XML translation of the IETF S/MIME ESS [11] security label. It has the following fields as defined in [11]:

- **Security Policy Identifier**: A security policy is a set of criteria for the provision of security services. It indicates the semantics of the other security label components.

- **Security Classification**: A Security Classification may have one of a hierarchical list of values defined by the security policy in force.
- **Privacy Mark**: The Privacy Mark may give additional required information defined by the security policy in force or by the originator of the security label.
- **Security Categories**: The Security Categories provide further granularity for the sensitivity of the information.

Each user is issued a certificate (X.509), which is extended to include her security privileges. A user in this context may be a person, a role, an application or a process. The users are granted security privileges, which are compared with the security labels of the objects, which the user requests access to. This may be UDDI records of the service registry or notifications, which the user has initiated a subscription for. An illustration of the relations between object security labels and user certificates is given in Figure 3.7.

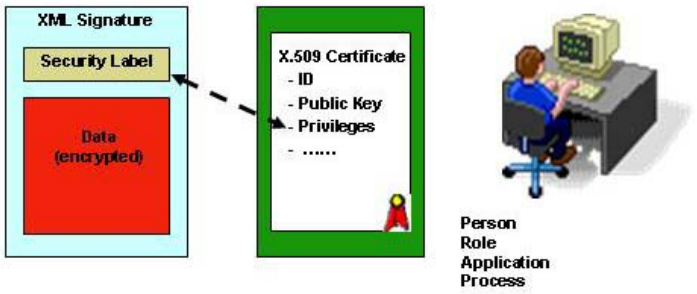

Figure 3.7 The Security Label bound to the information is compared with the privileges in the user's certificate in for access control to the information object

The security privileges component of the certificate is defined as a "security label", and is named a "privilege label" (see [10] for more details).

The use of XML Security Labels is proposed in [6]. How to associate security tokens with SOAP messages is specified in WS-Security 2004 [11]. Placing the privileges in the certificate is not a dynamic solution in that one would need to issue a new certificate in order to change a user's privileges. This solution is chosen for simplicity.

3.3.3 Securing the UDDI registry

All records stored in the UDDI registry are labelled and signed in order to indicate their sensitivity and to protect them from being changed during storage.

UDDI v.3 defines Application Program Interfaces (APIs) for access to the data within the service registry. Two of these are the Inquiry API, which is used for searching for records, and the Publish API, which is used for insertion and updates of records. In order to secure these interfaces and enforce differentiated access control on the stored records, we have introduced a security component called the System Protection Component (SPC) as part of the Security Abstraction Layer in front of the UDDI APIs. This security abstraction layer will perform the WSS related security processing of the SOAP messages (authentication, signature handling

and encryption), in addition to performing differentiated access control on the UDDI records based on the security labels of the UDDI records and the privileges in the user certificates.

Inquiry API

Access to the methods of the UDDI Inquiry API Set is restricted to users with a valid certificate, and the SOAP message carrying the inquiry needs to be correctly signed and encrypted. Access to the information in the result set of the inquiry is controlled comparing the security label of the UDDI records with the privileges in the user's certificate. The user certificate is retrieved from the distributed LDAP directory using the X509IssuerSerial and X509SubjectName from the signature of the incoming SOAP message. A list of UDDI records that matches the user's privileges is built and returned to the requestor in a compressed, encrypted, labelled and signed SOAP message. This process is illustrated in Figure 3.8.

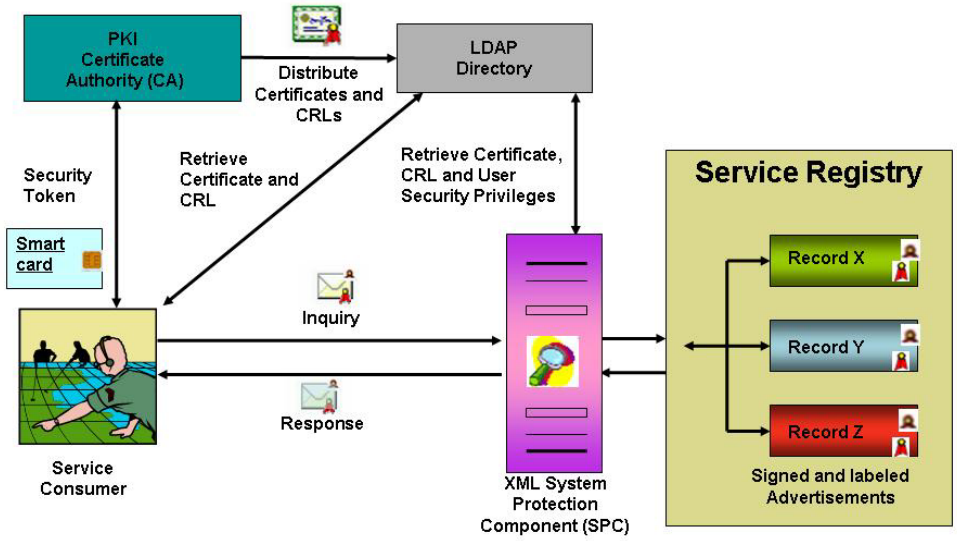

Figure 3.8 Securing the UDDI registry requires support of PKI and LDAP Directories for distribution of Certificates and CRLs

Publish API

In order to be allowed to publish to the registry, a publisher must be listed in the Access Control List of the registry. The publisher gets an authorization token by requesting the UDDI Security Policy API. This authorization token gives the right to publish using the UDDI registry Publication API. This functionality is a part of the software from Systinet [13].

SOAP messages carrying UDDI requests and responses must be labelled and signed correctly in order to be forwarded from the abstraction layer to the UDDI registry. Several services may be published in the same publish message given that they have equivalent security labels. If the services have different security labels, they must be published using one publish message for each variation of the security label. The security label from the SOAP message used in the publish request, will be used to mark the records put into the UDDI registry. The SOAP message used to send the response will have the same security label as was used for the request.

3.3.4 Securing Subscriptions and Notifications

The WS-BaseNotification standard [15] makes a distinction between the roles NotificationConsumer and Subscriber, but in this context a Subscriber will also be the Notification Consumer. These restrictions influence the specification of the security functionality because it is not allowed to subscribe to a service on behalf of others.

When a subscription request is received in a SOAP message, security processing of the SOAP message is performed (as described above). The X509IssuerSerial and X509SubjectName of the SOAP signature may be used to fetch the certificate with the User Privileges from the LDAP Directory. The NotificationProducer will create a Subscription Resource for the Subscription. The User Privileges found in the certificate will be included in this Subscription Resource for matching against the InformationSecurityLabel of the Notifications.

The NotificationProducer will match the InformationSecurityLabel in the SOAP message of the notification against the User Privileges registered for each subscription. A match is required to issue the Notification. Notifications will be encrypted, labelled and signed by the Notification Producer. The classification of the InformationLabel attached to the SOAP message will be set to the highest classification of the included information.

3.3.5 Architecture

The security concept described in this document results from our work with the group NATO RTO IST “Secure SOA Supporting NEC”, and is also in line with in the long term goal of the NATO NEC Feasibility Study [5], where the security is moved to the end systems.

This Security architecture describes a set of national LANs interconnected through the CWID WAN. A *Secured Web Services Gateway* is used in one of the domains for access control of the information entering and leaving the national domains. Each national LAN contains a *Web Service Provider* and a *Directory*. The services that a nation wants to share with its allies are replicated to the Web Service Provider of the LAN.

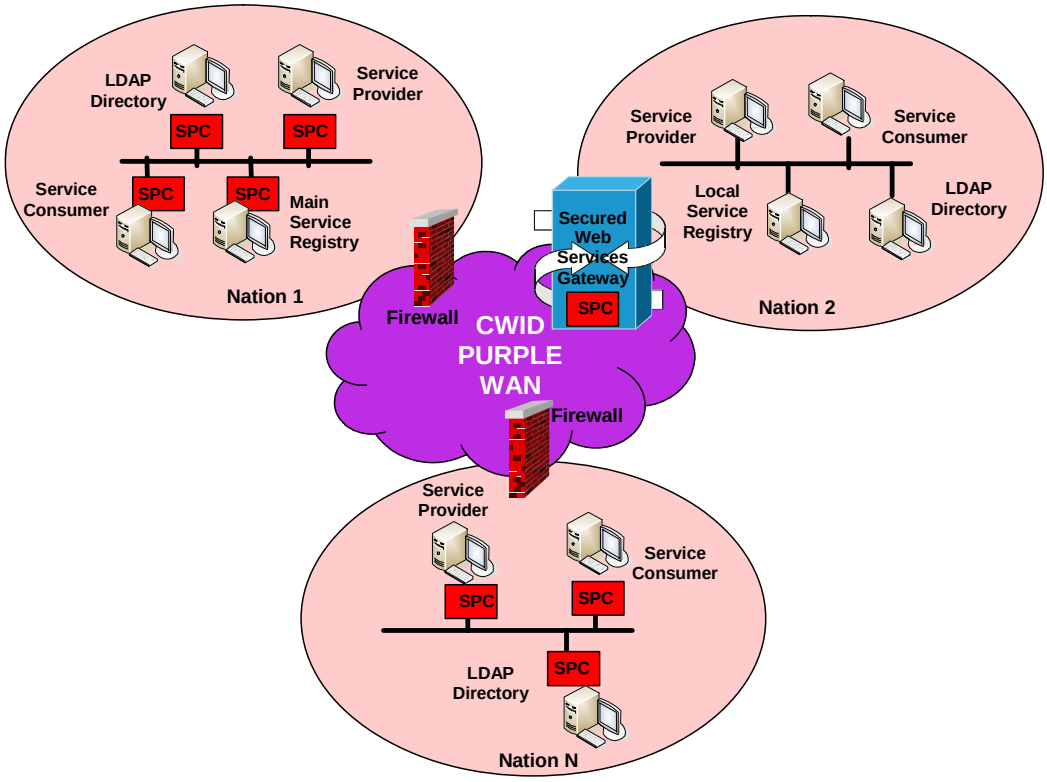

Figure 3.9 The figure shows the security model of the distributed demonstrator. The security functionality (SPC) may be placed in the end systems or in a Secured Web services Gateway.

In addition, one of the nations (or the NATO organization) will provide a Main *Web Service Registry* for looking up services published by the nations. Other nations may have a Local *Web Service Registry*, which may be synchronized with the Main service registry. The Directory systems are used for replication of X.509 Certificates and Certificate Revocation Lists (CRLs). The *System Protection Component*s (SPC) will provide the end-to-end security processing of the Web Services components.

Security Infrastructure

In order to use digital signatures and asymmetric encryption, a security infrastructure was needed in order to issue and distribute Certificates and Certificate Revocation Lists (CRLs). To serve these mechanisms we used a PKI system consisting of Certificate Authorities, Certificates, and LDAP servers. The commercially available KeyOne product from Safelayer [12] was used as CA and OpenLDAP [6] was used for directory services. Smartcards were also used to store user certificates.The replication of the LDAP information were done periodically exchanging LDIF files [11] using Publish/Subscribe functionality. By subscribing to the periodical update of each national LDAP server (using the WS-Notification specification), the LDAP information replicated will be protected by the SOAP security functionality. This also shows how a non-XML legacy system like LDAP may be included using Web Services technology.

All components that provide or consume services must contain security functionality. Figure 3.9 shows SPCs at all nodes that are not protected by the secure gateway. This component will handle all parts of the security processing, i.e., perform certificate validation, create and

validate signatures, encrypt and decrypt, and do access control based on the security labels. Thus, in our architecture security is handled in an end-to-end fashion.

3.4 Object-oriented C2IEDM

3.4.1 Theory

For data exchange on an interoperability level (i.e. between nations), we chose the data model defined by the Multilateral Interoperability Programme (MIP) [4]. This is an effort towards providing a common understanding of the battle space between different countries, and independent of doctrines, procedures, and tactics. The MIP model has been developed over many years of work, starting as a land model, and it is currently being extended to cover joint environments. The aim of the MIP is to achieve international interoperability of Command and Control Information Systems (C2IS) at all levels, in order to support multinational operations.

Note that, we are only using the data model from MIP and have chosen the C2 Information Exchange Data Model (C2IEDM) from MIP Baseline 2. Instead of using database replication as defined by MIP in the current Data Exchange Mechanism (DEM), we are using Web Services as the information exchange mechanism.

3.4.2 Architecture

In order to adapt the model to our needs, we have defined a suitable subset of the C2IEDM, which we call a *miniMIP* [6], and we exchange information using an object-oriented (OO) XML-version of this model. In the Entity-Relationship (ER) diagram for the original C2IEDM, there are approximately 240 entities. Using expertise on MIP and taking our internal data model into consideration, we selected 30 of these entities, sufficient to represent the information present in the internal model. Out of these 30 entities, there are six independent entities, i.e., entities that do not depend on other entities for identification. These are:

1) *object_item*, which represents an object, either materiel or organization;
2) *object_type*, which describes the type of an object item;
3) *affiliation*, which denotes the nationality of an object item;
4) *location*, which denotes the position of an object item together with
5) *vertical_distance*; and finally
6) *reporting_data*, which provides information (metadata) about reports. All other entities are dependent on one or more of these six entities.

The ER diagram of these 30 entities provides a good human-readable description of the information exchange contents, but such a representation is inherently tied to storage of information in a relational database. Therefore, it was necessary to transform this representation into something that was more suitable for message exchange. Given that the information to be exchanged was about physical objects present in the battlefield, our approach was to use object items as the fundamental entity, and then include all relevant data connected to that entity.

Using an object-oriented XML-version of the C2IEDM, the result is an object item XML structure containing all other relevant structures (embedding), which is one of two alternative

ways of structuring such an XML-document. The general rule in this approach is that everything, except other object items, can be embedded in an object item. Thus, for reporting organizations and for object item associations, object-id (OID) references must be used. This rule is necessary to avoid infinite loops.

It should be noted that there is also an alternative approach, where a flat structure is used. In this approach, all identifiable entities (the six entities listed above) are placed directly below the root (C2IEDM) element, and all associations are realized using OID references.

We chose to use the embedding approach for our demonstrator, as we believed that this structure would provide easier processing of the messages sent and received. Our experiences from the demonstrator showed that this was only partially true. On the one hand, the messages no doubt became more human-readable, with all information related to an object item grouped together. On the other hand, making every object item structure self-contained, including type, location, and affiliation, as well as all associated reporting data does introduce considerable redundancy, since much information will be repeated for every object item.

For instance, object type information, which could be common for several object items, is embedded within the object item, and therefore needs to be repeated. Type information can constitute as much as 28 lines of XML code, so it is clear that this principle of embedding can represent a considerable overhead. The problem is particularly pronounced for reporting data (i.e., information *about* a report): It is reasonable to assume that several pieces of information normally would be included in the same report (in other words, *one* report can contain information about several types of information on an object, such ass object type, affiliation, and status). However, the embedding principle means that the reporting data must be repeated for each such piece of information.

```
       <ObjectItemAffiliationInObjectItemList>
            <ObjectItemAffiliationInObjectItem>
                <Affiliation xsi:type="AffiliationGeopolitical"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                    <OID>209400000000003</OID>
                    <Code>NOR</Code>
                </Affiliation>
                <ObjectItemAffiliation>
                    <ReportingData xsi:type="ReportingDataAbsoluteTiming"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                        <OID>2090200000000000003</OID>
                        <CategoryCode>REP</CategoryCode>
                        <ReportingDate>19700101</ReportingDate>
                        <ReportingTime>000004</ReportingTime>
                        <ReportingOrganisationRef xsi:type="UnitRef">
                            <OID>209000000000123</OID>
                        </ReportingOrganisationRef>
                        <EntityCategoryCode>OIAFFL</EntityCategoryCode>
                        <EffectiveStartDate>19700101</EffectiveStartDate>
                    </ReportingData>
                </ObjectItemAffiliation>
            </ObjectItemAffiliationInObjectItem>
        </ObjectItemAffiliationInObjectItemList>
```

Figure 3.10 Example of the complexity of the miniMIP

Furthermore, the C2IEDM (and also the miniMIP) specifies that there may be multiple relationships between entities. This is realized by adding an index to the primary key in the association entities (e.g., in object-item-type, object-item-affiliation, and object-item-association), such that the index number separates the different relationships. Since the XML schema for the miniMIP is auto-generated, the result is a more complex schema, in order to handle such multiple relationships. As an example, in order to express the geopolitical affiliation of an object item (a three-letter code), six lines of XML are needed, and together with the reporting data, a total of 21 lines of XML are needed, as illustrated in Figure 3.10.

Another effect of embedding and multi-relationships is that the XML documents usually contain a relatively large number of nesting levels. Each document usually contains seven levels of sub-elements under the root element, and as we will describe later, this represents a challenge for the XML parsers.

4 EXPERIMENT RESULTS

This section contains the perceived results of the experiment. Technical results are first given for each of the four focus areas, and then there is a more general summary of overall results.

4.1 Service Discovery

The use of UDDI as service registry in this experiment is all in all satisfactory, although some points of improvement have been identified. It is important to note that the backend UDDI registry provided by Systinet worked as expected. The improvement potential identified is rather concerned with the UDDI specification rather than this implementation.

Actually, considerable amount of time was used in the design process just defining the metadata to be used, not on the implementation as such. Describing both services and business entities is important to be able to discover services, and more precisely the correct services, at the correct time. Coming to an agreement on what metadata was needed and how to represent these was a challenging task. And as experience shows, changes to the metadata had to be performed late in the process, as new or modified requirements emerged. UDDI is highly extendable when using the tModel construct, but it comes with a cost in complexity. In our limited experiment with simplified service and business descriptions, we had to produce a large number of tModels. This may lead to a management challenge when more complex environments are introduced. The time used in the design process substantiates the concern on complexity.

The ability to do service discovery based on geographical position or coverage area was identified as a very desirable feature for military service discovery. By nature these types of data are highly dynamic and it thus becomes a challenge to represent these values in a UDDI registry. This use is inconsistent with both the purpose and the design of UDDI, which is best suited to describe fairly stable services with stable descriptions. If this information is to be kept up to date at all times one would possibly encounter performance issues on both client and server side. Our solution to this problem involved defining a static coverage area for a service, and defining an additional service used to fetch the current position. This solution reduces the stress on the UDDI registry, but one should investigate the possibility of more general solutions for service discovery in a highly dynamic environment.

The adoption of the abstraction layer tier architecture, and the fact that we needed to extend the UDDI specification, should be enough to point out areas were we feel UDDI have shortcomings. The extra tier architecture has advantages, but also some disadvantages. Most notably is the danger of increasing response time for clients when one extra layer of processing is introduced. One factor playing an important role in this is the filtering. As mentioned above, filtering of information is performed for security reasons, geographical search and the identification of expired services. The problem is that filtering have to be performed on the reply from the backend UDDI server. Many UDDI inquiries return only partial information, and often the returned information is not sufficient to do filtering so the abstraction layer must fetch extra information. These numerous interactions with the backend UDDI registry may

become a performance issue. The extra functionality should ideally be placed within the UDDI registry to avoid these issues. However, since no open-source alternatives were present at the time of the experiment, this was not possible.

Another issue identified during the experiment is the problem of identifying one unambiguous security context for UDDI entities. The problem is that a UDDI entity often is put together of many small entities, which also can be used by other entities. This makes it difficult to label the objects. As a consequence it becomes difficult to establish one unambiguous security context to perform the object level access control on UDDI entities.

From the experiment described in this document it should be clear that creating an architecture for dynamic service discovery is hard. UDDI proved to perform as expected, but the need to include extra functionality does in itself prove potential for improvement. We would like to see the extra functionality included in UDDI, as this is becoming the de-facto service registry standard for Web Services.

Even though UDDI registries can be federated to provide a distributed registry, it is still a centralized architecture for service discovery. Decentralized discovery, known from peer-to-peer systems and others, may often be more appropriate in highly dynamic environments. In the future we would like to see a combination of these technologies. One scenario is using decentralized discovery to locate a more capable registry, which can be based on UDDI.

One of the key challenges of dynamic service discovery is closing the gap between design time and run-time discovery. In order to get a truly dynamic service discovery these two types of service discovery come closer. From our point of view this would involve using semantics and defining a common vocabulary for enabling the extended use of metadata. The ultimate goal would be to enable run-time discovery of new and previously unknown services during run-time. When using UDDI services are discovered during design-time and instances of these services can be discovered during run-time.

4.2 Publish/Subscribe

Having used the publish/subscribe pattern realized with WS-Notifications in a military context, we have gained much experience with this technology, presented and discussed in this section.

Publish/subscribe proved to be a reliable way of communication in our scenario. It provided us with a standardized interaction mechanism, which was a considerable step towards interoperability between the national C2 systems. All exchange of military intelligence data was based on this pattern, using different data formats for different data types.

In the subscription creation phase, the subscriber will suggest a termination time of the subscription. The publisher either accepts this, or decides a new termination time based on its local policies. At the later stage, it is possible to modify the lifetime of the subscription by sending a Renew or Unsubscribe message to publisher. After the initial subscription phase, a client will receive notifications from the publisher (NotificationProducer) as long as the subscription is valid.

However, a client has no means of controlling the size, amount, and frequency of notifications to receive. If the NotificationProducer generates notifications frequently, the NotificationConsumers may get flooded with large amount of messages. Consequences are increased CPU processing time, memory, and bandwidth usage. Although this was not a serious problem at CWID where a high speed network was available, this is an important issue to address when considering an operational implementation, where both computational and network resources may be limited. In such cases, there is a need to establish a policy that determines the size, amount, frequency and other Quality of Service (QoS) parameters between the NotificationConsumer and the NotificationProducer.

The WS-Notification specification does not define a way to set up QoS parameters per subscription - the flow control and QoS mechanisms are missing. However, the specification does offer an optional field called *SubscriptionPolicy*, which may be included in the subscription request message (see Figure 4.1). The content of the field is not specified, i.e. it is defined as an XML *Any* type, meaning that applications are free to use the *SubscriptionPolicy* field proprietarily, at the expense of interoperability.

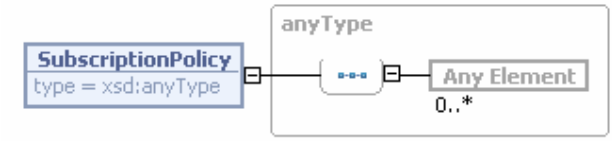

Figure 4.1 WS-Notification defines SubscriptionPolicy field without defining its content. SubscriptionPolicy could be used specify the lacking QoS between producer and consumers, but it needs to be standardized throughout the coalition.

In order to use the SubscriptionPolicy field for specifying the QoS parameters of military applications using the publish/subscribe mechanism, we need to define a common understanding, i.e. standardize the field content throughout the coalition. The SubscriptionPolicy parameters need to be specified in an XML schema and integrated into NotificationProducer implementations. Then, for each incoming subscription request, the content of the SubscriptionPolicy field would be validated against the SubscriptionPolicy XML schema. If the validation succeeds, the NotificationProducer would accept and store the requested QoS settings. If the validation fails, it would reject the subscription request.

We consider the following SubscriptionPolicy parameters to be necessary:

- *Message size*. Specifies the maximum size of the notification. Useful if the client has limited communication bandwidth or processing power
- *Message frequency*. Specifies whether the notifications will arrive asynchronously, or periodically. If periodically: specify the time period
- *Message content.* Specifies whether the message contains the “full dump” of the operational picture, or only the updates since the last notification.

The publish/subscribe mechanism was a reliable communication mechanism in our demonstrator. However, the challenges may become more obvious if the number of subscribers grows significantly larger than in our system. The message distribution is a potential bottleneck since web services utilize point-to-point communication, and more efficient mechanisms such as multicast of SOAP messages need to be considered.

Furthermore, the large size of notification messages could be reduced by transferring the full C2IEDM data model initially, and subsequently only transferring the updates since the last notification. Finally, various methods could be applied to reduce the overhead of XML data structures, such as binary XML and data compression.

WS-Notification supports hierarchical aggregation of topics into topic trees. We have omitted using this feature, leaving our topics flat – each topic representing all messages from a single service. However, topic trees could be introduced as a fine-grained filtering of messages to receive. For example, several subtopics could be defined for the ACP_MaritimePictureTopic:

- Based on unit classification: a subtopic called “Frigate” would deliver tracks for all observed frigate units
- Based on location: a subtopic could be defined to deliver maritime tracks for a specified geographical area

4.3 End-to-end Security

While implementing the security mechanisms, two distinct packages were identified; the Security Protection Component (SPC) and the Label Handling Component (LHC). The SPC is a generic component for signing and encrypting SOAP messages and it is implemented using various COTS software available from Apache and standard Java APIs for certificate handling. The LHC is a special purpose component developed for generating and comparing security labels for access control. This is also implemented using standard Java XML handling software.

The most significant challenge we experienced during the implementation of these security mechanisms was integration with the chosen COTS products, both for publish subscribe message exchange and the service registry. As a result of the choice of using the Systinet UDDI registry as our service registry, the Abstraction Layer had to include, and make use of, both the SPC and LHC. To enable access control to and ensure the integrity of the UDDI content, all records must be labelled and signed. Since records in UDDI often are comprised of numerous small entities with only loose connections, e.g., service descriptions with associated tModels, it becomes hard to establish one security context to label and sign. In our demonstrator we chose to only label business and service entities, since tModels often can be shared. To minimize the changes that had to be made to standard UDDI v3 client implementations, we chose to extract the security label associated with the SOAP messages when storing records in UDDI. This was possible since the content of these messages is identical to the records to be stored in the UDDI and should thus be classified at the same level under the same security policy.

The Access Control to the UDDI records is also performed by the Abstraction Layer at the Inquiry API. This includes checking the security label of the record against the user’s privileges and verifying the signature to ensure that the record has not been tampered with. Again, as with filtering of e.g., service expiration, the need to perform numerous interactions with the backend registry in order to retrieve enough information may reduce the performance of the Abstraction Layer (see section 3.1).

The actual integration of the SPC and LHC with UDDI Abstraction Layer proved to be one of the major challenges faced. Although the Abstraction Layer uses Apache Tomcat and Axis [1], which enabled us to do low level SOAP message manipulation, differences in the serialization of Java objects to actual XML documents often resulted in broken signatures. The lesson learned is that care has to be taken in order to preserve the signatures.

Integrating the SPC and LHC with the Globus Toolkit used for WS-Notification also proved to be a challenge. In order to ensure that no subscribers are receiving messages that they are not authorized for, all outgoing SOAP messages must be filtered. This is based on the fact that on time of subscription it is not guaranteed which security level the produced messages on a given Topic will have, and this may even change during execution. As a result, the Globus Toolkit Manager must store the privileges, or a link to the NotificationConsumers certificate, in order to do the matching between the XML security label of the SOAP message and the privileges. Furthermore, the SOAP messages must be encrypted and signed in addition to the fact that Globus Toolkit only provides access to high-level data structures and not the actual SOAP message. While this provides an easy to use interface to WS-Notification developers, it is a challenge when wanting to manipulate the actual SOAP message. It was solved by extending the Globus Toolkit source code to include the filtering mechanisms.

The specification did not include a secure binding between the certificate ID (Distinguished Name) and the FROM address in the SOAP message. This means that we couldn’t check if the FROM address was correct as part of the security verification. One solution to solve this could be to include the URL in the endpoint of the certificate.

4.4 Object-oriented C2IEDM

Although the miniMIP is small compared to the original C2IEDM, it is still a quite complex model, with its 30 different entities and a large number of relationships. Furthermore, using XML to express object-oriented structures inherently leads to relatively large and complex documents. Thus, during the development of the demonstrator, it quickly became clear that the complexity of the data model represented a considerable challenge for the participants. Substantial effort was required to achieve a common understanding of the model among the participants. An additional factor contributing to the complexity was the fact that, in several cases, we were unable to detect incompleteness of the C2IEDM documents during internal testing. It was first when exchanging data with our NATO partners that the errors became visible.

In order to help understanding the schema, and for testing out software, there was a particular need for example documents at all parties. However, without the necessary serialization software in place, the first examples had to be hand-made. Although containing several errors (in particular with respect to namespaces), these hand-made examples proved valuable as a basis for discussion during the early phase of the development process.

One complicating factor was the fact that the attribute EntityCategoryCode, under reporting data, is defined as mandatory. This attribute only exists in the physical schema, and not in the

logical schema. However, although not carrying information that is being used in the demonstrator, this attribute proved necessary, as the serialization of miniMIP-objects failed without it.

In addition, we found it necessary to make a few minor changes to the C2IEDM schema, in order to make the serialization/de-serialization work, and to produce valid XML documents. The most important change was the need to change the type defined for the C2IEDM element. In the original miniMIP schema, this element is defined as an anonymous type (i.e., a type with no name). However, this resulted in JAXB[4] not generating a marshalling class for the C2IEDM element, making it impossible to serialize miniMIP messages at all. Therefore, the type of this element was explicitly named C2IEDM, i.e., the same as the name of the element. This had no practical implications, and there were no changes in the produced XML documents. Furthermore, the arm-category-code under Unit-Type contained an empty value in its enumeration of allowed values. For some reason, the inclusion of this empty value caused the corresponding Java class for this attribute not to be generated, and we therefore removed this empty value.

As a result of the format of our internal COP, we made a clear distinction between reporting units and reported objects. Only the reported objects were displayed on our DPNs; and through an “own report”, there was an implicit association between a reporting unit and a corresponding physical object displayed on the DPNs. Furthermore, there is not much emphasis put on the reporting unit itself, beyond a name and some information about the sensor used.

In the miniMIP on the other hand, this association is made explicit, through the object-item-association entity. Furthermore, the miniMIP puts Materiel and Unit on the same footing (both are sub-types of Object-Item), which means that considerably more information is stored about Units. To resolve this imbalance, we chose a simple solution for our demonstrator, and maintained the approach used in our COP. This meant that relatively sparse information was provided for the units. In addition, the units were not associated with Materiel objects, which in turn meant that the units did not have a location.

Since location is not mandatory according to the C2IEDM schema, this was not a problem, although it was remarked by some of our partners. On the other hand, this also implied that we did not use the object-item-association construct, meaning that we did not get the chance to test this aspect of the miniMIP model.

It should also be mentioned that, in the COP model a track is identified by a trackId. Consequently, it is a prerequisite for the translator that the messages being translated from miniMIP are referentially complete, as the object-oriented COP model does not contain identifiers for other objects than tracks.

[4] Java Architecture for XML Binding, see Internet java.sun.com for more info

A final issue that caused some problems was the use of OIDs. One of our partners did not use OIDs internally, and therefore did not support maintaining OIDs between messages. As a consequence, the OIDs were not kept consistent between messages:

- Each object item may have different OIDs from message to message
- An OID used for one object item in one message may be used for another object item in the next message

The rationale for doing so was that the partner assumed a “cancel-update” approach. This means that each new message cancels and replaces the previous message, which in turn implies that when a new message arrived, all displayed object should be removed, and the new ones displayed instead.

This could potentially have represented a problem for us, since our DPNs are not able to remove objects from the display, and each new message therefore would lead to a new set of objects being displayed. However, our partner did not achieve the goal of sending periodic messages to us during CWID, so this did not become a problem. In addition, the partner used unique names for each of the object items, so it could have been possible for us to implement a mapping between names and OIDs.

4.5 Results summary

The results of the CWID 2006 SecSOA experiment are several. Depending on what stakeholders we refer to, different results can be identified.

First, for those who participated from the FFI-project ”NBF Beslutningsstøtte”, the experiment took a lot of efforts, especially if you include all preparation work. And the results of those efforts can be summed up in an extremely valuable learning experience for the participants. The specific technological results for each technical area are described in the previous subsections.

At the conceptual level, the results can be viewed as good examples of how SOA using Web Services may be a suitable technology for systems that are to support Network Based Defence or Network Enabled Capability. Military resources are made available as services, securely accessible from the network using end-to-end security. Services are described by metadata that is published to the network using a service registry.

Internationally, this experiment is a result of the work in the NATO Research Task Group IST-061, where all member nations have put in resources and efforts, and gained experience accordingly. An important result of the work in the group is the set of specifications [6] that has been developed. The viability of the experimental implementations of the specification is a reinforcement to the value of the specifications. An updated version of the specification document will be made publicly available. For further evaluations at the NATO RTO level, please refer to the final report from IST-061 that will be issued by the end of 2006.

Another international result is the fact that topics from this experiment have been presented as papers at CCRTS in June 2006 [11] and ICCRTS in September 2006 [3].

On the national level, information exchange between our state-of-the-art experimental Demonstrator and the national operational C2 system NORCCIS-II is a small result worth mentioning. Working together on-site CWID 2006 has reinforced relations between the participants of the Norwegian CWID delegation, creating a good foundation upon which an even better CWID participation for the coming years can be built. As CWID hosts for at least 2007 and 2008, Norway should aim for excellence in its own CWID participation.

Also, the SecSOA experience helps FFI and the Norwegian Defence in the evaluation of leading technology and how to use it in future implementations of the Information Infrastructure (INI).

For FFI, the results can be summed up into experience for the scientists and documentation aimed at internal and external use. FFI is in this experiment recognized as a valuable contributor to NATO RTO. Being clearly visible as a CWID 2006 participant is also assumed to be a positive result for FFI.

Finally, there are explicit results of the CWID Test Cases described in section 2. The NATO CWID 2006 Report concludes on the Norwegian SecSOA at an overall level: ”The information was retrieved successfully by partners.”

However, there are a few areas where our initial expectations were not met. Although the ambition level of the CWID Test Cases referred in section 2.2 was relatively high, it is necessary to point out that only four out of seven were considered 100% successful.

First, as indicated in section 2.2, ”Dynamic service replacement” (TC# 621) was cancelled. That was a team decision based on the fact that the implementation architecture did not provide fully dynamic behaviour in itself. Custom implementation would have been possible for demo purposes, but that was determined to be outside scope for the experiment.

Second, ”New services made ready for use” (TC# 616) was not tested with a “not previously defined data format”. Again, a customized demo could have been set up, but time and resources were used elsewhere. And finally, regarding “Access control at the object level” (TC# 619), the implementation of security on the service registry turned out not to be compatible with the respective implementation already made in the Data Publishing Nodes. As a result, we could not fully test object level security in the service registry.

These are to a large extent implementation shortcomings, and not limitations of the technical concept given in the specifications. But some of them clearly point in the direction of limited “dynamicity”, especially when it comes to the Service Discovery and the use of metadata.

It is hard to be objective about the level of success for an effort like the SecSOA experiment at CWID 2006. Many important goals have been achieved, but in hindsight it is easy to spot parts that could have been improved. On the overall, the many positive achievements certainly qualify for the label “Success”. Nevertheless, there are goals that were not met. “*Limited dynamicity*” and “*Specifications were not detailed enough*” are examples of limiting factors.

These examples may indicate that the appropriate label is “Partial success”, if we consider the shortcomings as reductions to the success level. On the other hand, it may be argued that these findings, and the learning process that lead to them, are very valuable results in itself. Several potential technological improvements have been identified. In that respect the shortcomings may count positively instead of limiting the success.

5 CONCLUSION

The experiment was successful in proving that the specifications could be implemented, and that actual information exchange took place between respective nations’ experimental systems. The interoperability testing during NATO CWID 2006 was purely technical, using a very simple simulated operational environment as demonstration backdrop. Participating teams gained valuable technical experience within each of the technological focus areas.

One lesson learned is that this kind of work demands lots of resources. Specifications development within leading edge technology areas is hard to get right first time. Errors and inconsistencies will be identified at implementation time, generating extra workload.

In the context of CWID it should be noted that interoperability testing requires partners in other nations. Experimental systems with newly developed interfaces will either have to ensure that parallel work is done in other nations – like we did, bringing our own partners – or make information exchange go through established national interfaces.

The SecSOA results from CWID 2006 give good examples of how SOA using Web Services may be a suitable technology for systems that are to support NBD. Military resources are made available as services, accessible from the network. Services are described by metadata that is published on the network. The results achieved clearly indicate that SOA is a good foundation for the future Information Infrastructure (INI). SOA has the potential of overcoming the limitations of current “stove-piped” solutions.

An example of achieved interoperability may be the fact that the SecSOA experiment was able to integrate two test-partners external to the IST-061 group, namely Spain and NC3A. They came in late in the process, chose to implement selected parts of the specifications and were able to interoperate with the rest of the group during CWID.

In the security area, SecSOA has initiated important work. End-to-end security at the object level is an important contribution to existing security regimes. It is an interesting future solution with a great potential, given adequate security policy and management procedures.

Our final conclusion must be that the experiment was a valuable experience for the participants, and the technologies look promising. We recommend the research work on these topics to be pursued.

References

[1] Apache Axis (2006): http://ws.apache.org/axis

[2] ebXML Registry (2006): http://www.oasis-open.org/committees/regrep/

[3] Haakseth R, Hadzic D, Lund K, Eggen A, Rasmussen R E (2006): Experiences from implementing dynamic and secure Web Services, Proceedings of the 2006 International Command and Control Research and Technology Symposium, Cambridge UK, 2006

[4] Multilateral Interoperability Programme (MIP): http://www.mip-site.org

[5] NATO Network Enabled Capability Feasibility Study, Volume 2

[6] NATO RTO IST-061: The NATO RTO/IST-061 Secure SOA Demonstrator Specification for CWID 2006, version 1.0 issued March 17, 2006 (may be accessed by contacting the participants of the group, may also be published as a future FFI/Notat)

[7] OASIS UDDI Version 3 Specification (2006): http://www.oasis-open.org/specs/index.php#uddiv3

[8] OASIS Using WSDL in a UDDI Registry, Version 2.0.2, Technical Note (2004). http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.htm

[9] OASIS Web Services Notification (WSN) Task Committee: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[10] Rasmussen R, Eggen A, Hadzic D, Haakseth R, Lund K, Rose K (2006): Experiment Documentation: ”Secure SOA supporting NEC” – NATO CWID 2006, FFI/NOTAT-2006/02539, ugradert (to be published)

[11] Rasmussen R E, Eggen A and Haakseth R(2006): An architecture for experimenting with secure and dynamic Web Services, Proceedings of the 2006 Command and Control Research and Technology Symposium, San Diego, USA, 2006.

[12] Safelayer: http://www.safelayer.com/

[13] Systinet: http://www.systinet.com/

[14] Web Services Dynamic Discovery (WS-Discovery) (2006): http://msdn.microsoft.com/ws/2004/10/ws-discovery/

[15] WS-BaseNotification 1.2 OASIS Standard: http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf

[16] WS-Security: SOAP Message Security: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[17] WS-Topics 1.2 OASIS Standard: http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf